CMSC330 Spring 2009 Midterm #2 Solutions

1. (12 pts) Programming Languages and Automata
    a. (4 pts) Explain why type inference works poorly with weak type systems

Weak type systems allow one type to be treated as another type (or performs implicit type casts) so it is much harder to determine the type of a variable based on how it is used in the program.

    b. (4 pts) Explain why upwards funargs are needed for currying

Currying allows a function to consume an argument, then returns an upwards funarg (function return value) to consume additional arguments.

Currying doesn't work if functions can't return functions, since the returned function is needed to consume additional parameters.

    c. (4 pts) Sketch two approaches we can use to prove context-free grammars are strictly more powerful than regular expressions.

2 pts for each (max 4)

1) **Convert a regular expression into a regular grammar**
2) **Convert a DFA into a regular grammar**
3) **CFGs are implemented using a NFA+stack. NFA alone can implement REs.**

2. (23 pts) OCaml Types and Type Inference
    a. (3 pts each) Give the type of the following OCaml expressions
        i. let f x y = (x, y) **Type = 'a -> 'b -> 'a * 'b**
        ii. let f (x , y) = x y **Type = ('a -> 'b) * 'a -> 'b**
        iii. let f x y z = z (y::x) **Type = 'a list -> 'a -> ('a list -> 'b) -> 'b**

Note that 'a -> 'b -> 'c and 'a -> ('b -> 'c) are identical, since -> is right associative. So the solutions above may have extra parentheses but still be correct.

    b. (4 pts each) Write an OCaml expression with the following type
        i. 'a -> int -> 'b -> int **Code = let f x y z = y+1**
        ii. 'a -> ('a -> 'b) -> 'b **Code = let f x y = y x**

Many possible answers, check in OCaml interpreter

    c. (3 pts each) Give the value of the following OCaml expressions. If an error exists, describe the error.
        i. let x y = fun z -> (x z) in let x = z in y **Value = error, unbound x**
        ii. let a = 1 in let f x y = x+y+a in f 7 8 **Value = 16**

3. (18 pts) OCaml Programming

Consider the OCaml type *bst* implementing a binary search tree:

```
type bst =
   Empty
   | Node of int * bst * bst ;;

let rec map f t = …                    (* type =  (int -> int) -> bst -> bst      *)
let add1 t =                           (* type = bst -> bst                       *)
```

a. (12 pts) Implement a function *map f t* that returns a tree with *f* applied to the int value of each node in tree *t*.

```
let rec map f n = match n with
   Empty -> Empty
 | Node (m, left, right) -> Node(f m, map f left, map f right)
```

// 2 pts for **match n with Empty -> Empty**
// 2 pts for **match n with Node (…)**
// 2 pts for **using Node constructor Node (x, y, z)**
// 2 pts each for **3 parts of Node constructor**
f m
map f left
map f right

b. (6 pts) Using *map* and an anonymous function (no helper functions) write a function *add1 t* that returns a tree nearly identical to tree *t*, but with the value of each node increased by 1.

```
let add1 t = map (fun x -> x+1) t
```

// 3 pts partial credit if used helper function instead

```
let add1_helper x = x+1
let add1 t = map add1_helper t
```

4. (13 pts) Context Free Grammars

a. (3 pts) Write a grammar for $a^x b^y$, where $x \leq y \leq 3x$, for $x,y \geq 0$

S → aSb | aSbb | aSbbb | epsilon

Given the following grammar

S → S ^ T | T
T → V # T | V
V → id

b. (2 pts) Which operator has higher precedence? **#**
c. (2 pts) Which operator(s) is right associative? **#**

d. (6 pts) Rewrite the following grammar so it can be parsed by a predictive parser

S → S ^ T | S # T | T
T → id

S → T L **// 2 pts**
L → ^ T L | # T L | epsilon **// 4 pts**
T → id

5. (16 pts) Parsing
Consider the following grammar: S → Ab | d A → aA | epsilon

a. (6 pts) Compute First sets for S and A
First(S) = First(Ab) U First(d) = {a,b} U {d} = {a,b,d} **// 4 pts**
First(A) = { a, epsilon } **// 2 pts**

b. (10 pts) Write a predictive, recursive descent parser for the grammar

```
parse_S( ) {
        if ((lookahead == "a") ||
            (lookahead == "b")) {      // S → Ab          2 pts
                parse_A( );
                match("b");
        }
        else if (lookahead == "d") {  // S → d           2 pts
                match("d");
        }
        else error( );                // error           2 pts
}
parse_A( ) {
        if (lookahead == "a") {       // A → aA          2 pts
                match("a");
                parse_A( );
        }
        else ;                        // A → epsilon     2 pts
}
```

6. (10 pts) Scoping & Lazy Evaluation

Consider the following OCaml code.

```
let app f y = let x = 2 in f y ;;
let add x y = let incr x = x+y in app incr (x+5) ;;
 (add 3 4) ;;
```

a. (2 pts) What value is returned by (add 3 4) with static scoping? Explain.

12. The sequences of calls & resulting values bound to the formal parameters is:
. add (x=3,y=4) calls app (f=incr, y=x+5=8) calls incr (x=8)
In the body of incr y is free and refers to the y in add x y (y=4), leading to 8+4=12

b. (4 pts) What value is returned by (add 3 4) with dynamic scoping? Explain.

16. The sequences of calls & resulting values bound to the formal parameters is:
. add (x=3,y=4) calls app (f=incr, y=x+5=8) calls incr (x=8)
In the body of incr y is free and refers to the y in app f y (y=8), leading to 8+8=16

c. (4 pts) Rewrite the following code (using thunks) so that the result is the same as if OCaml used call-by-name, even though OCaml uses call-by-value.

```
let f x = x+1 ;;  f y ;;
```

```
let f x = (x ( )) + 1 ;;          // 2 pts      apply x ( )
f (fun ( ) -> y) ;;               // 2 pts      put y in fun ( ) -> y
```

7. (8 pts) Parameter Passing

Consider the following C code.

```
void swap(int f, int g) {   int tmp = f;    f = g;    g = tmp; }
int main( ) {
   int i = 2;
        int a[] = {2, 0, 1};
   swap(i, a[i]);
   printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
}
```

a. (2 pts) Give the output if C uses call-by-value
2 2 0 1 // (no effect).

b. (2 pts) Give the output if C uses call-by-reference
1 2 0 2 // (swap i & a[2])

c. (4 pts) Give the output if C uses call-by-name
1 2 2 1 // (i = 2, tmp = i, i = a[i], a[i] = tmp)

Answers must be exactly correct for credit