Midterm Exam, COMP3031 L1, Fall 2012

Date	Nov 2, 2012 (Friday)
Time	19:00-21:00
Instructions:	(a) This exam contains six problems, counting for a total of 100 points.
	(b) Write ALL answers in the exam book. Do not use any other papers.

Name:	Problem	Points
Student ID:	1.	
ITSC Account:	2.	
	3.	
	4.	
	5.	
	6.	
	Total:	

Problem 1 (15 pts) What is the value of each of the following SML expressions (a)-(c)?

```
(*a*)
fun ack(0, n) = n+1
   | ack(m, 0) = ack(m-1, 1)
   | ack(m, n)= ack(m-1, ack(m, n-1));


(*b*)
fun merge L [] = []|
    merge [] L = L |
    merge (x::xs) (y::ys) = if(x>y) then (merge xs ys) else (y::(merge xs ys));


(*c*)
datatype tree = nil | node of int * tree * tree;

fun ts(v1, nil) = 0
  | ts(v1, node(v, nil, nil))= if v > v1 then v else 0
  | ts(v1, node(v, L, R)) = if v > v1 then v + ts(v1, L) + ts(v1, R)
      else  ts(v1, L) + ts(v1, R);
```

(a)

```
ack(1,2);
```

(b)

```
merge [1,3,5,7] [8,6,4,2];
```

(c)

```
ts(2, node(3, node(1, nil, nil), node(4, node(5,nil,nil), node(2, nil, nil))));
```

Problem 2 (15 pts) What is the type of each of the following SML functions (a)-(c)?

(a)

```
(*foo1*)
fun foo1 f v = not (f v);
```

(b)

```
(*foo2*)
fun foo2 f [] v = v |
    foo2 f (h::t) v = f h (foo2 f t v);
```

(c)

```
(*foo3*)
fun foo3 x y z w [] = (y, z)|
    foo3 x y z w (h::t) =
       if x(h,w)
       then foo3 x (h::y) z w t
       else foo3 x y (h::z) w t;
```

Problem 3 (20 pts) Write the following SML functions (a)-(d) on `datatype tree = nil | node of int * tree * tree`.

(a) `exist(v,t) = fn : int * tree -> bool`. Given an integer `v` and a tree `t`, return true if there is a node in `t` whose integer label equals `v`;otherwise, return `false`. e.g., `exist(2, node(2,nil,nil))` returns `true`.

(b) `insert(v,t) = fn : int * tree -> tree`. Given an integer `v` and a tree `t`, insert `node(v,nil,nil)` into `t` by the following recursive rule: if `t` is `nil`, return `node(v,nil,nil)`; otherwise, if `v` > `t`'s integer label, insert `node(v,nil,nil)` into `t`'s right subtree; otherwise, insert `node(v,nil,nil)` into `t`'s left subtree. E.g., `insert(4, node(2, node(1,nil,nil), node(3,nil,nil)))` returns `node (2,node (1,nil,nil),node (3,nil,node(4,nil,nil)))`.

(c) `traverse(t) = fn : tree -> int list`. If `t` is nil, return `[]`; otherwise, return a list of integers from traversing the left subtree appended with the list of the integer label of the tree appended with the integer list from traversing the right subtree. e.g.,
`traverse(node (2,node (1,nil,nil),node (3,nil,node(4,nil,nil))))` returns `[1,2,3,4]`.

(d) `search(v,t) = fn : int * tree -> int list`. If `t` is nil, the function returns `[]`; otherwise, it returns a list of integers that are the integer labels of the nodes along the search path. The search path is from the root to the first node with the integer label `v` if such node exists in the tree; otherwise, it is from the root to a leaf node. The list is generated as following: if a node's integer label equals `v`, return [v]; otherwise, if `v` is greater than the node's integer label, return the list of the node's integer label appended with the list generated from search in the right subtree; otherwise, return the list of the node's integer label appended with the list generated from search in the left subtree. e.g., `search(5, node (2,node (1,nil,nil),node (3,nil,node(4,nil,nil))))` returns `[2,3,4]`. search(1, node (2,node (1,nil,nil),node (3,nil,node(4,nil,nil)))) returns `[2,1]`.

Problem 4 (15 pts) Consider the following grammar in BNF with `<S>` being the starting non-terminal:

```
<S>::= a<S>b<S> | b<S>a<S> | <empty>
```

(a) Generate all **unique** strings of length less than 6 in the language represented by this grammar.

(b) Determine whether the language this grammar generates is the same as one generated by `(a(ab|ba)*b(ab|ba)*|b(ab|ba)*a(ab|ba)*)*`. If your answer is yes, just say so and no explanation is needed. If your answer is no, give an example string that belongs to one language but not the other.

Problem 5 (15 pts) We define the following language of set expressions

1. 'A', 'B', 'C', 'D', and 'E' are set expressions.
2. X $\cup$ Y, X $\cap$ Y, and X $-$ Y are all set expressions, given both X and Y are set expressions.

In decreasing order, the precedence of the $\cap$, $\cup$, and $-$ operators is as follows

1. $\cap$ (left associative)
2. $\cup$ (left associative)
3. - (left associative)

(a) Write an **unambiguous** context-free grammar in BNF for these set expressions. Your grammar should conform to the given operator precedence and associativity rules.

(b) Draw the **tree representation** for the evaluation of "A-B$\cup$C$\cap$D-E".

Problem 6 (20 pts) Consider the following simplified SML type system:

- "bool", "int", "real", and "string" are SML types.
- An SML type followed by "list" is an SML type, e.g., "int list".
- An SML type A followed by "− >" followed by an SML type B (A and B can be the same or different) is an SML type, e.g., "int− >int".
- An SML type A followed by "*" followed by an SML type B (A and B can be the same or different) is an SML type, e.g,, "int*string".
- An SML type surrounded by a pair of parentheses is an SML type, e.g., "(int*string)"

The composition of SML types obey the following rules in decreasing precedence:

```
()
list
*     (left associative)
->    (right associative)
```

(a) Write an **unambiguous** context-free grammar in BNF for these SML types preserving the precedence and associativity of type composition.

(b) Draw the parse tree of "`string list -> (string -> int) -> (string * int) list`" according to your grammar in (a).

Blank Page