Cornell University
Preliminary Exam 2 — 13 Nov 2007

Solutions

1. True/False [16 pts]

   (parts a–h; 1 point off for each wrong answer, 0 points for each blank answer)

   (a) A declaration `val empty :'a stack` in the signature of a stack data structure indicates that the implementation is functional (i.e., the stack is immutable).

   True

   (b) Representation invariants should be documented in module interfaces.

   False

   (c) During each garbage collection phase, a copying collector traverses exactly half of the memory.

   False

   (d) In an AVL tree, the difference in length between the shortest and the longest path from the root to the leaves is at most one.

   False

   (e) Hash table operations are performed in constant time if the number of buckets is proportional to the size of the table and the hashing function satisfies the uniform hashing assumption.

   True

   (f) In functional languages that use function closures to model higher-order functions and static scoping, it may not be safe to pop function frames off the stack as functions return.

   True

   (g) The amortized complexity of an insertion operation into a resizable array with n elements is $O(lg\ n)$.

   False

   (h) Union-find data structures can unify arbitrarily large equivalence classes in $O(1)$ time.

   True

2. References [16 pts] (parts a–b)

(a) [4 pts] Write a short ML program whose execution produces memory cells that would not be reclaimed by a reference counting garbage collector. Briefly explain.

Answer:

```
datatype mlist = Nil | Cons of mlist ref
let val x = ref Nil
    val y = Cons x
in
    x := y
end
```

The datatype `mlist` *defines mutable linked lists, and the code essentially creates a cyclic list with one element. More precisely, the code creates a reference cell* `x` *and a cons cell* `y`*, each of them pointing the other. Even after the let block finishes and* `x` *and* `y` *go out of scope, the cells still have non-zero reference counts. Hence a reference counting collector will not collect them.*

(b) [12 pts] Right-threaded trees are binary trees where each unused right pointer from a node to a Nil child is replaced with a pointer to the node's in-order successor. This makes it easy to traverse the entire tree by following right links.

Consider the following datatype for binary trees with mutable links:

```
datatype 'a tree = Nil | Node of 'a * 'a tree ref * 'a tree ref
```

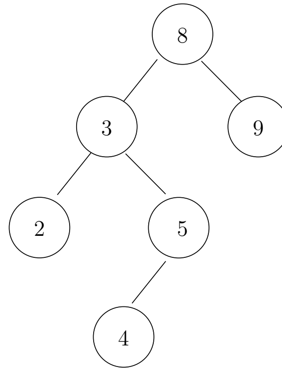
Finish up the implementation of the function `rthread` below that updates a tree, building all of its right-treaded links. The function updates the existing tree without building any new tree nodes. It returns `SOME x` where `x` is the right link of the rightmost tree node, if any; or `NONE` otherwise.

Answer:

```
fun rthread(t : 'a tree) : 'a tree ref option =
    case t
     of Nil => NONE
      | Node (_, l, r) =>
            (case (rthread (!l), rthread  (!r)) of
                 (NONE, NONE) => SOME r
               | (NONE, y) => y
               | (SOME x, NONE) => (x := t; SOME r)
               | (SOME x, y) => (x := t; y))
```

3. Balanced Trees [16 pts] (parts a–d)

(a) [3 pts] Consider the following binary search tree:



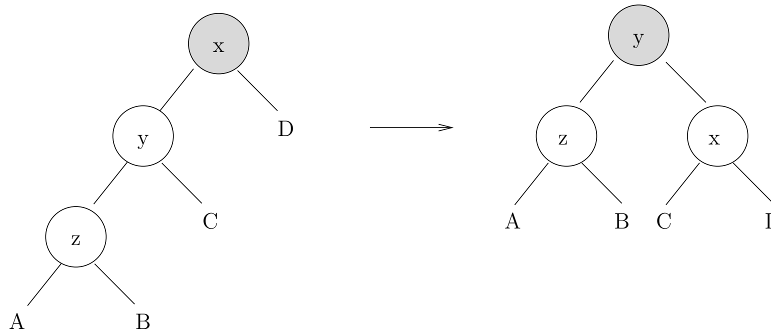
Can this be an AVL tree? Show why or explain why not.

Answer: *No. The height difference between the two subtrees of the root is 2.*

(b) [3 pts] Can the above tree be a red-black tree? Show why or explain why not.

Answer: *Yes. Coloring nodes 2, 5, 8, and 9 black; and nodes 3 and 4 red makes the resulting tree a valid red-black tree.*

(c) [6 pts] Consider the operation described in the figure below, where a rotation is used to re-balance a sequence of two red nodes in a red-black tree. In this figure, x, y and z are single nodes, and A, B, C, and D are subtrees. Shaded nodes are black, and clear nodes are red. Nodes x and y are re-colored by this operation.



Is this operation always safe? If yes, argue why. If not, describe the problem, the cases when it happens, and give a solution.

Answer: *This rotation incorrect if D has a red root: the path* $y - z - D$ *in the resulting tree will have two consecutive red nodes.*

We can fix this by instead coloring z and x black and coloring y red. The resulting subtree will always be a valid red-black subtree, regardless of the color of D's root.

(d) [4 pts] Explain why the above re-balancing operation would be desirable.

Answer: *The operation would be useful because the resulting subtree has a black root, hence it can't violate the red-black invariant of its parent. As a result, no*

subsequent rotations would propagate up the tree. The rebalancing process stops at this point.

In contrast, a rebalancing operation that produces a subtree whose root is red might break the parent's invariant. Thus, more rotations might propagate up the tree.

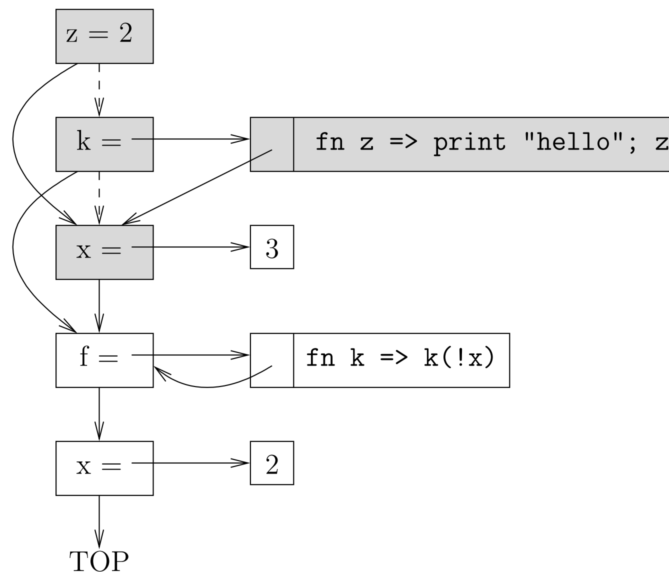
4. Environment Model [20 pts] (parts a–b)

(a) [15 pts] Consider the following program:

```
let val x = ref 1
    fun f(k) = k (!x)
    val x = ref (x := 2; 3)
in
    f (fn z => (print "hello"; z));
    (x, f)
end
```

Draw the environment diagram at the point where the program prints `"hello"`. Clearly indicate the current environment, as well as the environments saved at each function call.

Answer:



(b) [5 pts] What memory cells that can be collected at the end of this computation?

Answer: *The shaded boxes in the figure above become unreachable and can be collected when the entire let block finishes.*

5. Types [22 pts] (parts a–c)

(a) [6 pts] Write a function `f` with the following type:

```
val f:  (('a ref) * 'b -> 'c) -> ('a -> 'b -> ('c ref))
```

Answer: *One example:*

```
fun f g = fn a => fn b => ref (g(ref a, b))
```

(b) [4 pts] Write a one-line function for which ML cannot infer a valid type. Briefly explain.

Answer: *There are many possible answers. Here's one:*

```
fun f x = x x
```

(c) [12 pts] Consider the following SML function for which we want to infer an appropriate type:

```
fun g(x,y,z) = if y 1 then x z else (y, z 2)
```

Write down the set of type constraints for function `g`. For each equation, indicate how it has been derived. Then show the solution types inferred for each of the arguments `x`, `y`, and `z`.

Answer: *Assign fresh type variables to* `g`*'s parameters and return value:*

```
x:'a
y:'b
z:'c
g:'a*'b*'c->'d
```

Build the type constraints:

`'b = int->bool`	*because of expression* `y 1` *in the test.*
`'a = 'c->'e`	*because of expression* `x z` *on the true branch.*
`'c = int->'f`	*because of expression* `z 2` *on the false branch.*
`'e = 'b*'f`	*because both branches must have the same type.*
`'e = 'd`	*because* `g`*'s return type must match the type of* `g`*'s body.*

Solution:

```
x:(int->'f)->((int->bool)*'f)
y:int->bool
z:int->'f
g:(int->'f)->((int->bool)*'f) * (int->bool) * (int->'f) -> (int->bool)*'f
```

6. Recurrences [10 pts] (parts a–b)

Consider the following recurrence relation:

$$T(1) = 1$$
$$T(n) = 3T(n/2)$$

(a) [5 pts] Show that $T(n)$ is $O(n^2)$.

Answer:
Let $P(n) = T(n) \leq n^2$.
Claim: $P(n)$ *holds for all* $n \geq 1$.

Proof by strong induction on n*:*
Base Case: $T(1) = 1 \leq 1^2$.
Indution Hypothesis: Assume that $P(n)$ *holds for all* $k < n$.
Inductive Step: Prove that $P(n)$ *holds. We have:*

$$\begin{aligned} T(n) &= 3T(\tfrac{n}{2}) \\ &\leq 3(\tfrac{n}{2})^2 \quad \textit{by IH} \\ &= \tfrac{3}{4}(n^2) \\ &\leq n^2 \end{aligned}$$

Since $T(n) \leq n^2$*, by the definition of Big-O notation it means that* $T(n)$ *is* $O(n^2)$.

(b) [5 pts] Find a constant c such that $T(n)$ is $\Theta(n^c)$. Explain.

Answer: *Try to find constant* c *such that* $T(n) = n^c$ *for all* $n \geq 1$*. We plug this expression into the recurrences and get:*

$$\begin{aligned} 1^c &= 1 \\ n^c &= 3(n/2)^c \end{aligned}$$

The first equation is trivially true. From the second equation we get $c = \log_2(3)$.