CS312 Preliminary Exam 2
Cornell University, Spring 2005 April 19, 2005

Solutions

1. Short Answers [20 pts] (parts a–e)

   For each of the questions below, provide a short answer (1-2 sentences). Ambiguous answers will receive no points.

   (a) [4 pts] Consider an stack abstract data type that provides an `empty()` method to create an empty stack. In order to save space, we'd like to implement `empty()` such that it returns the same heap object every time it is called. When is this approach safe, and when it is not? Explain why.

   Answer: *It is safe for functional implementations where data structures are immutable, but is unsafe for implementations using imperative data structures.*

   (b) [4 pts] What is the amortized run-time complexity of inserting n elements into a standard hash table if the table size is increased by a fixed amount each time it becomes full, and each resizing requires re-hashing all of the elements currently present in the table?

   Answer: $O(n^2)$. *With increments of 1 and a load factor of 1, re-hashing is done at each insertion, and the total re-hashing work is:* $1 + 2 + ... + n = O(n^2)$.

   (c) [4 pts] Why is it better to use variable-sized blocks for memory management, instead of fixed-size blocks?

   Answer: *It avoids internal fragmentation, i.e., wasting space inside each block.*

   (d) [4 pts] Red-black trees are binary search trees that are approximately balanced. Briefly explain what "approximately balanced" means.

   Answer: *The longest path from the root to the leaves is at most twice as long as the shortest path from the root to the leaves.*

   (e) [4 pts] Describe the main disadvantages of garbage collection via reference counting, compared to the other garbage collection techniques.

   Answer: *It incurrs run-time overhead for each heap update (read or write). And it cannot collect leaked cycles.*

2. Zardoz Refs [10 pts] (parts a–b)

   For the following programs, fill in the box with an expression that causes each program to evaluate to 42.

   (a) [4 pts]

```
let
  val f = ref (fn x => x + 1)
  val zardoz = [          ]
in
  (!f)(40)
end
```

Answer: `f := (fn _ => 42)`

(b) [6 pts]

```
let
  datatype node = NULL | PTR of node ref
  fun f(h:node, c:node):int =
    case c of
      NULL => raise Fail "No cycle."
    | PTR p => if !p = h then 1
               else 1 + f(h, !p)
  val zardoz:node = [          ]
in
  f(zardoz, zardoz) * 21
end
```

Answer:

```
let val n:node ref = ref NULL
in (n := PTR(ref(PTR(n))); !n)
end
```

3. Complexity [25 pts] (parts a–e)

Consider a datatype that describes tree structures. We assume that nodes do not carry values, since those are not relevant to this problem.

```
datatype tree = Leaf | Node of tree * tree
```

We can define a function that computes the height of a tree as follows:

```
fun height(t:tree):int =
  case t of
    Leaf => 0
  | Node(t1,t2) => 1 + Int.max(height(t1), height(t2))
```

Let $T(n)$ be the running time of function `height` for a tree with n nodes and leaves.

(a) [3 pts] Write a formula $T(1)$ for a tree with one leaf ($n = 1$), and a recurrence relation for $T(n)$, for a tree with $n > 1$ nodes and leaves. Use constants c_1, c_2, etc. for pieces of computation that are $O(1)$.

$$\begin{aligned} T(1) &= c_1 \\ T(n) &= T(n_1) + T(n_2) + c_2, \quad \text{where } n_1 + n_2 = n - 1 \end{aligned}$$

(b) [6 pts] Find a solution for your recurrence relation, and prove your result by induction. Make sure you specify what kind of induction you use.

Answer: *The solution is* $T(n) = (c_1 + c_2)n - c_2$. *We prove this fact using strong induction on* n, *for* $n \geq 1$.

Base case $n = 1$. *Then* $T(1) = c_1 = (c_1 + c_2) \cdot 1 - c_2$.

Assume that $T(m) = (c_1 + c_2)m - c_2$ *for all* $m \geq 1, m \leq n$.

Induction step: we shall prove that $T(n+1) = (c_1 + c_2)(n+1) - c_2$. *We know that:*

$$T(n+1) = T(n_1) + T(n_2) + c_2$$

where $n_1 + n_2 = n$. *Therefore,* $n_1 \leq n$ *and* $n_2 \leq n$. *By Induction Hypothesis for* n_1 *and* n_2 *we get:*

$$\begin{aligned} T(n_1) &= (c_1 + c_2)n_1 - c_2 \\ T(n_2) &= (c_1 + c_2)n_2 - c_2 \end{aligned}$$

Therefore:

$$\begin{aligned} T(n+1) &= T(n_1) + T(n_2) + c_2 \\ &= (c_1 + c_2)n_1 - c_2 + (c_1 + c_2)n_2 - c_2 + c_2 \\ &= (c_1 + c_2)(n_1 + n_2) - c_2 \\ &= (c_1 + c_2)(n+1) - c_2 \end{aligned}$$

A tree is *balanced* if, for each node in the tree, the height difference of its children is less than 1. The function below uses function `height` to determine whether a tree is balanced:

```
fun balanced(t:tree):bool =
  case t of
    Leaf => true
  | Node(t1,t2) =>
      balanced(t1) andalso balanced(t2) andalso
      (abs(height(t1) - height(t2)) <= 1)
```

(c) [3 pts] Write a recurrence relation for the running time $B(n)$ of function `balanced`, for a tree with n nodes and leaves. For simplicity, assume that the tree is complete, so that all of the leaves are at the same depth. Use constants c_i for pieces of computation that are $O(1)$.

$$B(n) = 2B(\tfrac{n-1}{2}) + 2T(\tfrac{n-1}{2}) + c_3 = 2B(\tfrac{n-1}{2}) + c_4 n + c_5$$

(d) [3 pts] What is the run-time complexity of function `balanced`? You don't have to prove the formula.

Answer: $O(n \log n)$.

(e) [10 pts] Write another function `balanced2` that has better run-time complexity than function `balanced`.

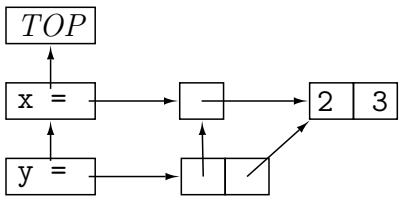
```
fun balanced2(t:tree):bool =
  let fun balanced'(t:tree):bool*int =
       case t of
         Leaf => (true,0)
       | Node(t1,t2) => let
           val (b1,h1) = balanced'(t1)
           val (b2,h2) = balanced'(t2)
         in
           ((abs(h1-h2) <= 1) andalso  b1 andalso b2,
            Int.max(h1,h2) + 1)
         end
  in
    #1 (balanced' t)
  end
```

4. The Environment Model [25 pts] (parts a–c)

(a) [5 pts] Draw the environment diagram for the following declarations:

```
val x = ref (2, 3)
val y = (x, !x)
```

Answer:



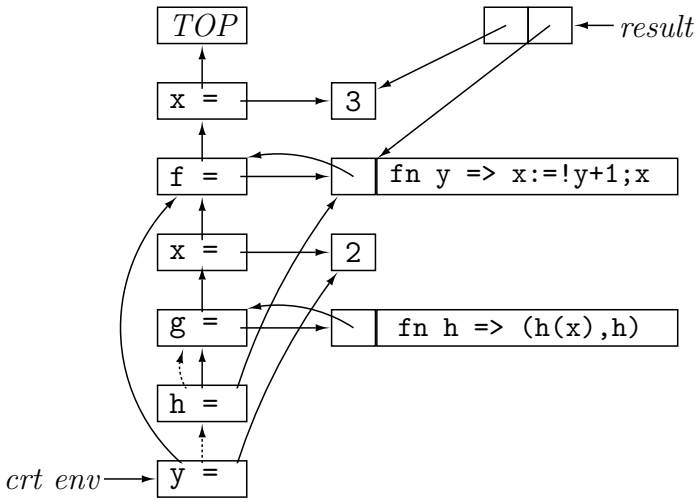
(b) [15 pts] Now consider a more complicated program:

```
let val x = ref 1
    fun f(y) = ( x := !y + 1; x )
    val x = ref (!x + 1)
    fun g(h) = (h(x), h)
in
    g(f)
end
```

Assume that the language allows recursive function definitions, so the environment pointer in the closure of a function `f` points to the environment that defines `f`.

Draw the environment produced by the evaluation of the above program. Indicate the result value and the current environment right before variables start being popped off the environment.

Answer:



(c) [5 pts] Indicate the heap cells can be reclaimed by the garbage collector at the end of the program from part (b).

Answer: *Among the heap objects, only the reference cell containing integer 2, and the closure for function* `g` *can be collected.*

5. Types [20 pts] (parts a–c)

(a) [5 pts] Match each of the following expressions with their corresponding types.

```
1. fn x => fn (y,z) => y          A. 'a -> 'b -> ('b * 'a)
2. fn x => fn y => (y,x)          B. 'a -> ('a -> 'b) -> 'b
3. fn (x,y) => fn z => y          C. 'a -> ('b * 'c) -> 'b
4. fn x => fn y => fn z => y      D. ('a * 'b) -> 'c -> 'b
5. fn x => fn y => y x            E. 'a -> 'b -> 'c -> 'b
```

Answer: *1C, 2A, 3D, 4E, 5B.*

(b) [5 pts] Write an SML program that would be free of run-time type errors if executed, but which fails to type-check (that is, SML issues a type error message). Do not use if expressions.

```
let val x = ref true
in
   (x := 0; !x + 1)
end
```

(c) [10 pts] Consider the following SML declarations where types have been omitted:

```
fun f(x,y,z) =
   (y x) x  orelse  z (x 1)
```

What is the type that SML infers for `x`, `y`, and `z`?

```
x : int -> 'a
y : (int -> 'a) -> (int -> 'a) -> bool
z : 'a -> bool
```