15-150 Spring 2012
Lab 6

February 22, 2012

1 Introduction

The goal for this lab is to make you more familiar with continuations, and proofs about continuations.Please take advantage of this opportunity to practice writing functions and proofs with the assistance of the TAs and your classmates. You are encouraged to collaborate with your classmates and to ask the TAs for help.

1.1 Getting Started

Update your clone of the `git` repository to get the files for this weeks lab as usual by running

```
git pull
```

from the top level directory (probably named `15150`).

1.2 Methodology

You must use the five step methodology for writing functions for every function you write on this assignment. In particular, every function you write should have a purpose and tests.

2 Proofs about Continuations

Consider the following two functions:

```
fun sum (t : int tree) : int =
    case t of
        Empty => 0
      | Leaf x => x
      | Node(t1,t2) => sum t1 + sum t2

fun sumc (t : int tree) (k : int -> int) : int =
    case t of
        Empty => k 0
      | Leaf x => k x
      | Node (l,r) => sumc l (fn a => sumc r (fn b => k (a + b)))
```

We would like to prove that the continuation-based `sumc` behaves the same as `sum`, when you apply `sumc` to the identity continuation:

Theorem 1. *For all* `t : int tree`*,* `sumc t (fn x => x)` $\cong$ `sum t`*.*

We can try to prove this theorem by induction on `t`.

Task 2.1 Start the case for `Node(l,r)`. Explain why the proof breaks down.

To fix this, we can generalize the theorem to consider an arbitrary continuation `k`:

Theorem 2. *For all values* `t : int tree`*,* `k : int -> int`*,*
`sumc t k` $\cong$ `k(sum t)`

This says that `sumc` computes the same sum as `sum`, and then passes this sum to `k`.

To combat the problem you encountered above, it is necessary to quantify over `k` *in the predicate that is proved by induction*, so that the inductive hypotheses are general enough.

Task 2.2 Prove "for all `t`, $P(\texttt{t})$" by induction on `t`, where P is defined by

$$P(\texttt{x}) = \text{for all } \texttt{k}, \texttt{sumc x k} \cong \texttt{k(sum x)}$$

You may assume that `sum` is total.
Case for `Empty`:
To show:

Proof: Assume a continuation `k`.

Case for `Leaf x`:
To show:

Proof:

Case for `Node(l,r)`:
IH 1:

IH 2:

To show:

Proof:

When you use an IH, carefully note which continuation the "for all" is instantiated with.

Have the TAs check your proof before continuing!

3 Programming with Continuations

3.1 Answer types

`sumc` can in fact be given a more general type than above. Starting from

```
fun sumc (t : int tree) (k : ?) : ? = <as above>
```

Task 3.1 Infer the most general type for `sumc`. Explain why this makes sense.

Task 3.2 Your answer should say that `sumc` is polymorphic, with one type variable. Give two example `k`s, which require instantiating the type variable to two different types.

3.2 Find

Task 3.3 Write a function

```
fun find (p : 'a -> bool) (t : 'a tree) : 'a option = ...
```

such that

- if there is some `x` in `t` for which `p x == true` then `find p t == SOME x`. If there is more than one `x` that satisfies `p`, return the left-most one in the tree.
- `find p t == NONE` if there is no such `x`.

Task 3.4 Write a function

```
fun find_cont (p : 'a -> bool) (t : 'a tree) (k : 'a option -> 'b) : 'b = ...
```

such that (1) `find_cont p t k` $\cong$ `k (find p t)` and (2) `find_cont` uses constant stack space.

Have the TAs check your code before leaving!