15-150 Fall 2011
Lab 6

October 5, 2011

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

Please download the `lab06.sml` file from the course Web site or copy it from the lab afs directory

```
/afs/andrew.cmu.edu/course/15/150/asgn/lab/06/
```

This file contains some of the datatype definitions and functions that were discussed in the last couple lectures.

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 (t1,t2) => sumc t1 (fn a => sumc t2 (fn b => k (a + b)))
```

Theorem 1. *For all values* `t : int tree`, `k : int -> int`, `n : int`,
if `sumc t k` $\Longrightarrow$ `n` *then*
there exists an `m:int` *such that* `sum t` $\Longrightarrow$ `m` *and* `k m` $\Longrightarrow$ `n`.

Case for `Empty`:
To show: for all `k : int -> int`, `n:int`, if `sumc Empty k` $\Longrightarrow$ `n` then
there exists an `m:int` such that `sum Empty` $\Longrightarrow$ `m` and `k m` $\Longrightarrow$ `n`.

Assume `k`,`n` such that `sumc Empty k` $\Longrightarrow$ `n`.
To show: there exists an `m` such that `sum Empty` $\Longrightarrow$ `m` and `k m` $\Longrightarrow$ `n`.

Take `m` to be 0. To show: `sum Empty` $\Longrightarrow$ `0` and `k 0` $\Longrightarrow$ `n`.
First, we show that `sum Empty` $\Longrightarrow$ `0`.
Proof: by calculation.

Next, we show that `k 0` $\Longrightarrow$ `n`. We will use determinism.

```
    sumc Empty k
|-> case Empty of Empty => k 0 | ...
|-> k 0
```

By assumption, `sumc Empty k` $\Longrightarrow$ `n`.
Thus, by determinism, `k 0` $\Longrightarrow$ `n`.

Case for `Leaf x`:
To show: for all `k : int -> int`, `n : int`, if _____

then there exists an `m:int` such that _____ and _____.

Proof: Assume `k`,`n` such that _____.

Need to show: there exists an `m` such that _____ and _____.

Proof: Take `m` to be _____.

First, observe that _____.

Second, we show that `k` _____ $\Longrightarrow$ `n`. We will use determinism.

```
    sumc ____________________________
```

```
|->
```

```
|-> k ___________
```

By assumption, `sumc` _____ $\Longrightarrow$ `n`.

Thus, by determinism, `k` _____ $\Longrightarrow$ `n`.

Case for `Node(t1,t2)`:
IH 1: for all `k1 : int -> int`, `n1 : int`, if _____

then there exists an `m1` such that _____ and _____.

IH 2: _____

To show: for all `k : int -> int`, `n : int`, if _____

then there exists an `m` such that _____ and _____

Proof: Assume `k`,`n` such that _____

Want to show: there exists an `m` such that _____ and _____

Proof: First, we will use determinism. Observe that

```
    sumc (Node(t1,t2)) k

|->

|->
```

By assumption, `sumc (Node(t1,t2)) k` $\Longrightarrow$ _____.

Thus, by determinism, (i)

Next, we will use IH 1 on fact (i), taking
`n1` to be _____

`k1` to be _____

The IH proves that there exists an `m1` such that _____

and _____

Next, using determinism,

```
(fn a => sumc t2 (fn b => k (a + b))) m1
```

```
|->
```

and by the IH `(fn a => sumc t2 (fn b => k (a + b))) m1` $\Longrightarrow$ _____.

Therefore, by determinism (ii) _____.

Next, we will apply by IH 2 to fact (ii), to show that

there exists an `m2` such that _____

and _____
Thus, there exists an `m` such that `sum (Node(t1,t2))` $\Longrightarrow$ `m` and `k m` $\Longrightarrow$ `n` because

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` and `k (find p t)` compute to the same value and (2) `find_cont` uses constant stack space.

Have the TAs check your code before leaving!