CSI 3120 Concepts of Programming Languages
Fall 2020

Assignment 4

Assigned November 4, due Sunday, November 15 at 11:00pm

1. (β-Reduction)

   Consider the following expression in the lambda calculus:

   $$(\lambda f.\lambda x.f\ x)\ (\lambda g.\lambda y.g\ y)\ (\lambda z.z+2)\ 3$$

   Each expressions of the form $(\lambda x.e_1)e_2$, which can be reduced using the β-reduction rule, is called a *β-redex*. Evaluate this expresssion by repeatedly applying the β-reduction rule until it can no longer be applied. Underline or circle the sub-expression that you are reducing at each step using β-reduction.

   Note: the above expression is written following the conventions of removing parentheses whenever possible; it is equivalent to the following one which includes all possible parentheses:

   $$\Big(\Big(\big((\lambda f.\lambda x.(f\ x))\ (\lambda g.\lambda y.(g\ y))\big)\ (\lambda z.(z+2))\Big)\ 3\Big).$$

2. (Substitution and Variable Renaming)

   Compute the result of the following substitution, renaming bound variables as necessary so that the required substitution is well-defined:

   $$[(z+y+1)/x]((\lambda y.\lambda z.(y-z)+x)\ x\ 0)$$

   Show the renaming and substitution steps. You don't have to apply any β-reduction steps.

3. (Parameter Passing)

Consider the following pseudo-Algol code.

```
function g(i, j)
  begin
    j := j + i + 2
    i := i * j;
    return i + j;
  end

  var x : int;
  var y : int;
  var z : int;
  var w : int;
  x := 5;
  y := 1;
  z := g(x,y);
  w := g(y,y);
  print x;
  print y;
  print z;
  print w;
```

(a) What numbers are printed when running the program using pass-by-value parameter passing?

(b) What numbers are printed when running the program using pass-by-reference parameter passing?

4. (Activation Records and Scope)

Consider the OCaml code below.

```
let z = 6 in
let f x =
  (let y = 4 in
   x + z + y) in
let z = 3 in
let y = f z in
y + z
```

This code contains 4 nested blocks, declaring `z`, `f`, `z`, `y`, followed by the body of the innermost block, which is the expression `y + z`. In (a) and (b) below, you are asked to show activation stacks for the execution of this code. Follow the requirements and guidelines below:

- In your activation records for calls to `f`, include all of the information that is shown on page 41 of the course notes for Mitchell, Chapter 7, except for the return address. You should include an intermediate result for the value of `x + z + y`. You may include other intermediate results if you would like, but they are optional.
- Recall that on page 38 of the course notes, it says "As before, each `let ... in ...` is a separate block, but only at the top level. Inside function definitions a series of `let ... in ...` declarations in parentheses will be considered local variables of the function." Note that function `f` has both a parameter and a local variable.
- The format of the activation records for declarations is on page 17 (but without the access link). Access links, control links and local variables are required in your activation records for declarations. There will be only one local variable in this kind of activation record, which is the name of the variable or function being defined. Intermediate results may be included, especially if there is a complex calculation required to determine the value of the variable being declared. For function declarations in this exercise, you do not have to give the value of the function (the closure). You can leave it blank.
- You are required to include the return-result address in your activation records for function calls. Many examples in the course notes do not show this information. See the recursive function examples on pages 25-32 and 58 for examples that do include this information.
- For an example involving dynamic scope, see page 39. Note that activation records for languages that use dynamic scope do not include an access link.

(a) Assuming static scope, show the activation stack for the execution of this code up to the point where the call to `f` completes execution, but the activation record for this call has not yet been popped off the stack.

(b) What is the value of this expression at the end of execution of this program (under static scope)?

(c) Consider your solution to part (a), ignore the access links, and now assume dynamic scope. What values in the activation stack change? You may describe and/or draw the changes to the stack.

(d) What is the value of this expression at the end of execution of this program under dynamic scope?

5. (Activation Records and Functions as Arguments)

Consider the OCaml code below.

```
let x = 10 in
let h (y:int) = 2*(x-y) in
let f (g:int->int) =
  (let x = 4 in
   g (3*x)) in
let x = 5 in
f h
```

Assuming static scope, draw the activation stack for the execution of this code starting with the first declaration of the variable `x` and ending with the activation record that is created when the function `h` is called inside the function `g`. Use the requirements and guidelines above with the following exceptions:

- You do not need to include a return-result address in function call blocks.
- Include values of functions in function declaration blocks. Recall that function values are represented by closures and that a closure is a pair consisting of an environment (pointer to an activation record) and the compiled code.
- Include the values of `(2*(x-y))` and `(g (3*x))` as intermediate results.