Homework 1: JavaScript and λ calculus

CSE 130: Programming Languages

Early deadline: August 8 23:59, Hard deadline: August 10 23:59

Names & IDs:

Notation

For this problem set we may use (and you may find it helpful to use) a less verbose syntax for λ-calculus programs. Specifically, instead of writing $\lambda x.(\lambda y.(\lambda z.e))$, we may drop the parentheses and write this term as $\lambda x.\lambda y.\ldots.\lambda z.e$. In general, the body of a λ abstraction extends as far right as possible. To further shorten notation, we may also drop the λ's in our example term and write it as $\lambda xyz.e$ when the variable names are obvious and not ambiguous.

Similarly, instead of writing $(e_1\ e_2)\ e_3$ we'll simply write $e_1\ e_2\ e_3$. We can drop parentheses in function applications since applications in λ-calculus are always assumed to be left-associative.

1 JavaScript hoisting and block-scoping [12pts]

Recall that JavaScript variable declarations using `var` can have one of two scopes: function scope or global scope. Variables declared inside a function have function scope while variables declared outside of functions have global scope. Regardless of where variables are actually declared and initialized, their declarations (but not initializations) are *hoisted* to the top of the nearest (function or global) scope; `var`s are not block scoped. This is different from `let` and `const` variables which are block scoped, as in most languages you might have encountered.

In this problem, you will implement block scoping for `var` declarations using first-class functions. In each part, you will be asked what the result of running a snippet of code is, and—to receive full credit—why. If the code fails with an error instead of returning a value, explain what caused the error.

1. [1pts] What does `f(x)` return? What values of `x` and `y` are used in the computation and why?

```
var x = 5;
function f(y) { return x + y; }
f(x);
```

Answer:

2. [1pts] What does `f(x)` return? What values of `x` and `y` are used in the computation and why?

```
var x = 5;
function f(y) { return x + y; }
if (true) {
  var x = 10;
}
f(x);
```

Answer:

3. [1pts] What does `f(z)` return? What values of `x` and `y` are used in the computation and why?

```
var x = 5;
function f(y) { return x + y; }
if (true) {
  var z = 20;
}
f(z);
```

Answer:

Block scoping as used in other languages can be emulated in JavaScript by creating an anonymous function and executing it immediately. In effect, each `(function () {` begins a new block because the body of each JavaScript function is in a separate block. Each `})();` closes the function body and calls the function immediately so that the function body is executed.

Joe vaguely remembered this technique from class and decided to implement a source-to-source compiler that added these anonymous functions around `var` definitions. Since Joe is clever, his compiler only wraps code where necessary—"why wrap things when you don't need to?" thought Joe. The results of his approach to the above snippets are given below.

4. [2pts] What does `f(x)` return? What values of `x` and `y` are used in the computation and why? Explain how this behavior differs from that of part (1) above.

```
(function () {
  var x = 5;
})();
function f(y) { return x + y; }
f(x);
```

Answer:

5. [2pts] What does `f(x)` return? What values of `x` and `y` are used in the computation and why? Explain how this behavior differs from that of part (2) above.

```
var x = 5;
function f(y) { return x + y; }
if(true) {
  (function() {
    var x = 10;
  })();
}
f(x);
```

Answer:

6. [2pts] What does `f(z)` return? What values of `x` and `y` are used in the computation and why? Explain how this behavior differs from that of part (3) above.

```
var x = 5;
function f(y) { return x + y; }
if(true) {
  (function () {
    var z = 20;
  })();
}
f(z);
```

Answer:

7. [3pts] Rene thinks that Joe's compiler may have been a bit too clever, leading it to behave incorrectly in some cases. Is she right? If so, explain where Joe's compiler got it wrong above and describe a general wrapping algorithm that Joe should have used instead.

Answer:

2 λ-calculus $\Leftrightarrow$ JavaScript [14pts]

To get you more comfortable with λ-calculus syntax, in this problem, we'll be converting λ-calculus expressions to JavaScript expressions and back. At first, you may find it helpful to think about λ expressions in terms of their JavaScript counterparts; but once your comfortable with the λ-calculus syntax you may find the converse to be true!

First, we'll convert some λ terms to JavaScript. You may use JavaScript arrow functions, but don't have to.

1. Convert $\lambda x f g h.(f\ x)\ (g\ x)\ (h\ x)$ to the equivalent JavaScript expression.

Answer:

2. Convert the following term to the equivalent JavaScript expression.

$$\begin{aligned}\lambda a_1 a_2 c_1 c_2.\quad & \textbf{if } a_1 + a_2 < 10 \\ & \textbf{then } c_1\ a_1\ a_2 \\ & \textbf{else } c_2\ a_2\ a_1\end{aligned}$$

Answer:

3. Convert $(\lambda x.y)((\lambda x.x\ x)(\lambda y.y\ z))$ to the equivalent JavaScript expression.

Answer:

Let's now convert from JavaScript to λ-calculus.

4. Write the JavaScript program `f(g(3))` and helper functions `f` and `g` as a single λ-calculus expression.

```
const f = (x) => x*2;
const g = function (y) { return y-1; }
f(g(3))
```

Answer:

3 λ-calculus and macro processors [10pts]

Macro processors, such as `cpp` and `m4`, do a form of program manipulation before the program is further compiled. You can think of this as symbolic program "pre-evaluation." For example, if a program contains the macro

```
#define square(x) ((x)*(x))
```

macro expansion of a statement containing `square(y+3)` will replace `square(y+3)` with `(y+3)*(y+3)`.

One way to build a macro processor is to associate a lambda expression with each macro definition, and then do β-reduction wherever the macro is used. For example, we can represent `square` as the lambda expression $\lambda x.(x * x)$ and β-reduce:

$$(\lambda x.(x * x))\ (y + 3) =_\beta (y + 3) * (y + 3)$$

if `square(y+3)` appears in the program. Some problems may arise, however, when variables that appear in macros also appear in the program.

1. Suppose a program contains three macros:

```
#define f(x) (x+x)
#define g(y) (y-2)
#define h(z) (f(g(z)))
```

Macro `h` can be written as the following lambda expression:

$$\lambda z.\ \underbrace{(\lambda x.x + x)}_{f}\ (\underbrace{(\lambda y.y - 2)}_{g}\ z)$$

Simplify the expression `h(3)` by using β-reduction. Do *not* simplify the arithmetic. Only reduce one step at a time. Use as many or as few lines as you need.

$$\begin{aligned} & (\lambda z.(\lambda x.x + x)\ ((\lambda y.y - 2)\ z))\, 3 \\ = \\ = \\ = \\ = \\ = \\ = \end{aligned}$$

2. A problem arises if a local variable inside a macro has the same name as an argument the macro is invoked with. Assuming that `typeof` is supported, the following macro works as long as neither of the parameters is named `temp`.

```
#define swap(a,b) { typeof(a) temp = a; a = b; b = temp; }
```

Show what happens if you macro-expand `swap(x, temp)` without doing anything special to recognize that `temp` is *bound* in the body of the macro. You do not need to convert this macro to lambda notation.

Answer:

3. Explain what the problem is and how you can solve it.

Answer:

4. Show how your solution would expand `swap(x, temp)` properly.

Answer:

4 Free Variables [6pts]

The FV function we discussed in class is a function that operates on *syntax*. (If fact, you can think of FV as one of the simplest possible *static analysis*.) Since FV is defined recursively, it must be *total*, i.e., it must be able to take any program (term) written in λ-calculus as input. To this end, the function is defined for every kind of term:

$$\begin{aligned} FV(x) &= x \\ FV(\lambda x.e) &= FV(e) \setminus \{x\} \\ FV(e_1\ e_2) &= FV(e_1) \cup FV(e_2) \end{aligned}$$

Given this, find the free variables in the terms below:

1. $FV(\lambda x.(\lambda y.y)) =$
2. $FV(\lambda x.(\lambda y.x)) =$
3. $FV((\lambda x.x\ y)(\lambda y.y\ x)) =$
4. $FV((\lambda p.\lambda q.\lambda r.p\ q\ r)(\lambda p.\lambda q.p\ q\ r)) =$
5. $FV(x\ y) =$
6. $FV(((\lambda f.\lambda y.\lambda x.f(y(x)))(\lambda x.y+z))(\lambda z.y-x)) =$

5 λ-calculus reduction [6pts]

In class, we talked about β-reduction (reduction via capture-avoiding substitution) and α-conversion (variable renaming). These two rules can be used to fully describe the semantics of λ-calculus and evaluate (a la term rewriting) any λ program.

There is, however, a third rule called η-conversion that is often used in compiler optimizations and, in Haskell, for point-free programming (as we will see). η-conversion says that you wrap any function term in a lambda expression: $\lambda x.e\ x =_\eta e$ if $x \notin FV(e)$. Since this is just another equation in our toolbox, you can also drop abstractions according to η-conversion.

In this problem you will reduce the following term $(\lambda x.\lambda y.x\ y)(\lambda x.x\ y)$ according to our equations theory. At every step note if you are doing a β-reduction, α-conversion, or η-conversion. You should do all possible reductions to get the shortest possible expression. There are multiple ways to reduce this expression; below you will consider two different ways, both starting with an α-conversion. You may **not** do multiple steps at once or use more lines than allocated.

First approach:

$$\begin{aligned} & & (\lambda x.\lambda y.x\ y)(\lambda x.x\ y) \\ (\alpha) & = & \\ (\quad) & = & \\ (\quad) & = & \\ (\quad) & = & \end{aligned}$$

Second approach:

$$
\begin{array}{rl}
 & (\lambda x.\lambda y.x\ y)(\lambda x.x\ y) \\
(\alpha) & = \\
(\quad) & = \\
(\quad) & =
\end{array}
$$

6 Reductions and variable capture [14pts]

Reduce the following lambda calculus expressions to normal form using the call-by-value evaluation strategy unless otherwise noted, showing all steps. Circle your final answer(s). If there are variables captured, give both the capture-avoiding and non-capture-avoiding reductions. You may also use η-conversion, when necessary.

1. [2pts] Reduce $(\lambda x.\ (\lambda y.\ x))\ (\lambda y.\ x)$

   Answer:

2. [2pts] Reduce $(\lambda x.\ (\lambda y.\ x))\ (\lambda y.\ y)$

   Answer:

3. [2pts] Reduce $(\lambda xyz.\ \lambda fgh.\ f\ x\ (g\ y)\ (h\ z))\ h\ (\lambda ab.\ a\ (g\ b))\ f$

   Answer:

4. [2pts] Reduce $(\lambda xy.\ z)\ z\ z$

Answer:

5. [2pts] Let

$$S = \lambda xyz.\ x\ z\ (y\ z)$$
$$K = \lambda xy.\ x$$
$$I = \lambda x.\ x$$

Reduce $S\ K\ S$ using either call-by-value or call-by-name (specify which one), and give your answer in terms of S, K, and/or I.

Answer:

Acknowledgements

Any acknowledgements, crediting external resources or people should be listed below.