Homework 2

Due 13 October

Handout 2
CS242: Autumn 2004
6 October

Reading

1. Read Chapter 2 on computability. This is pretty short.

2. Read Chapter 4. You do not need to try to understand the fixed-point operator (page 64 and page 65 up through the end of Example 4.3) unless you find this intriguing. (Some people do!)

Problems

1. Partial and Total Functions

For each of the following function definitions, give the graph of the function. Say whether this is a partial function or a total function on the integers. If the function is partial, say where the function is defined and undefined.

For example, the graph of $\texttt{f(x)} = \texttt{if x} > 0 \texttt{ then x} + 2 \texttt{ else x}/0$ is the set of ordered pairs $\{\langle x, x+2\rangle \mid x > 0\}$. This is a partial function. It is defined on all integers greater than 0 and undefined on integers less than or equal to 0.

Functions:

(a) $\texttt{f(x)} = \texttt{if x} * 2 > 40 \texttt{ then } 2 * \texttt{x else x} + 1/0$

(b) $\texttt{f(x)} = \texttt{if x} * \texttt{x} > 1 \texttt{ then } 5/\texttt{x else f(x} + 1)$

(c) $\texttt{f(x)} = \texttt{if x} \leq 5 \texttt{ then log(x} - 1) \texttt{ else f(x} + 2)$

2. Finding Shortest Programs

Some optimizers try to find the fastest possible code, and some try to find the shortest sequence of instructions. Code size is often important in practice, and critical for devices like smart cards and embedded processors that have limited memory.

Suppose you are given a Lisp function `shortest` that, given any syntactically correct function as an argument, produces the shortest equivalent function. When we say two Lisp functions are equivalent, we mean that they compute the same partial function. Among several equally short representations, `shortest` chooses the one which is first in lexicographic order.

For example, consider the function that adds one to its argument, twice:

```
(lambda (x) (+ (+ x 1) 1))
```

We might find that `(shortest (lambda (x) (+ (+ x 1) 1)))` evaluates to:

```
(lamba (a) (+ a 2))
```

Can you solve the halting problem using `shortest`? More specifically, can you write a program that takes a Lisp function `P` and an integer `n` as arguments, and then decides whether `(P n)` halts?

If you believe that the halting problem can be solved if you are given `shortest`, then explain your answer by describing how a program solving the halting problem would work. If you believe that the halting problem cannot be solved using `shortest`, then explain briefly why you think not.

3. Ambiguous Grammars

Grammars are ambiguous if more than one parse tree exists for the same expression. For many mathematical expressions, ambiguity can be resolved sensibly using precedence and associativity to determine the correct parse tree. However, there other kinds of expressions where ambiguity cannot be resolved as easily. Consider the following grammar for representing conditional statements:

```
<exp>  ::=  if <exp> then <exp>
          |  if <exp> then <exp> else <exp>
          |  0
          |  1
          |  2
          |  3
```

(a) Show two different parse trees for the expression

```
if 0 then if 1 then 2 else 3
```

(b) How would you disambiguate the grammar from part (a)? You can change the language if you wish by adding extra keywords. (Hint: how does the `bash` shell scripting language solve this problem?).

(c) The C programming language allows simple `if` statements of the form

```
if (<expression>)
  <statement>;
```

where ⟨`statement`⟩ is any single-line statement, including `if-else` statements. If the programmer wants more than one statement to be executed in the `if` block, he or she must enclose the statement in curly braces:

```
if (<expression>)
{
  <statement>;
  <statement>;
        .
        .
        .
  <statement>;
}
```

Perl, on the other hand, does not allow the simple `if`-statement allowed by C. In Perl, all statements in an `if` block must be wrapped in curly braces, even if there is only one statement. Why do you think Perl's designer decided on this convention? Note that Perl contains an `if-else` statement as well.

4. Lambda Calculus Reduction

Use lambda calculus reduction to find a shorter expression for $(\lambda p.\lambda q.\lambda r.p\ q\ r)(\lambda p.\lambda q.p\ q\ r)$. Begin by renaming bound variables. You should do all possible reductions to get the shortest possible expression. What goes wrong if you do not rename bound variables?

5. Symbolic Evaluation

The Lisp program fragment

```
(define f (lambda (x) (+ x x)))
(define g (lambda (y) (- y 2)))
(f (g 2))
```

can be written as the following lambda expression:

$$\left(\underbrace{(\lambda f.\lambda g.\, f\ (g\ 2))}_{\text{(f (g 2))}}\ \underbrace{(\lambda x.x+x)}_{f} \right)\ \underbrace{(\lambda y.y-2)}_{g}$$

Reduce the expression to a normal form in two different ways, as described below.

(a) Reduce the expression by choosing, at each step, the reduction that eliminates a λ as far to the *left* as possible.

(b) Reduce the expression by choosing, at each step, the reduction that eliminates a λ as far to the *right* as possible.

6. Semantics of Initialize-Before-Use

A nonstandard denotational semantics describing initialize-before-use analysis is presented in the text.

(a) What is the meaning

$$\mathcal{C}[\![\texttt{x:=0;y:=0;if x = y then z:=0 else w:=1}]\!](s_0)$$

in the state $s_0 = \lambda y \in \mathit{Variables}.\mathit{uninit}$? Show how to calculate your answer.

(b) Calculate the meaning

$$\mathcal{C}[\![\texttt{if x = y then z:=y else z:=w}]\!](s)$$

in state s with $s(x) = \mathit{init}$, $s(y) = \mathit{init}$, and $s(v) = \mathit{uninit}$ or every other variable v.

7. Modifying functional programs

Quicksort is a well-known sorting algorithm. The algorithm works by choosing some element of the list to be sorted, called the *pivot*, and then splitting the list into elements less than the pivot and elements greater than the pivot. (Elements equal to the pivot could be placed in either list, as long as they are only counted once.) Then each sublist is sorted and the results concatenated. While the worst-case behavior of Quicksort is not very good, the average behavior is excellent. Some implementations use the first element in the list as the pivot, some choose the pivot randomly. Here are implementations of Quicksort in Haskell, a pure functional language, and C, an impure and not really functional language.

The first five lines below are the Haskell program. The first line says that `qsort` of the empty list is the empty list. In the second line, the form `qsort (x:xs)` indicates `qsort` applied to a nonempty list of the form `x:xs`, where `:` is the Haskell infix operator for list `cons`. In Lisp terms, `x:xs` is a nonempty list whose `car` is `x` and `cdr` is `xs`. Lines 2–5 of the code "say" that Quicksort of a nonempty list `x:xs` returns a list obtained by concatenating the Quicksort of the elements less than `x` with `x` itself and the Quicksort of the elements greater than or equal to `x`. Questions about the programs appear after the code.

```
qsort []     = []
qsort (x:xs) = qsort elts_lt_x ++ [x] ++ qsort elts_greq_x
                 where
                   elts_lt_x   = [y | y <- xs, y < x]
                   elts_greq_x = [y | y <- xs, y >= x]
```

```
qsort( a, lo, hi ) int a[], hi, lo;
{
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);

    t = a[l];
    a[l] = a[hi];
    a[hi] = t;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

(a) Which list element is used as the pivot in each example?

(b) Which code seems easier to understand? Why? What if you had never programmed in either language before?

(c) Explain how you might modify the Haskell program to use a randomly chosen element as the pivot. You do not need to write the Haskell code. Just explain what changes would be required. Pretend that you are working with someone who knows Haskell well, but doesn't know a thing about sorting lists. Think about the kind of instructions you would give this person to modify the code.

(d) Explain how you might modify the C code to use a randomly chosen list element as the pivot.

(e) Which code modification seems easier to do?

(f) Suppose you had to explain the modified code to an Army General who is going to use this in a system to control launch of nuclear missiles. The Army General has never taken a programming course, or perhaps knows a little Cobol. In which case would it be easier to convince a skeptic that the code is actually correct?

(g) How much extra space (memory), in addition to the space used to represent the input list, do you think the Haskell program uses? Since we haven't talked about how Haskell is implemented, just try to make a reasonable, educated estimate. How does this compare to the memory requirements of the C program?