DUE: Wednesday 1 March in class

Reading

See class web page

1. Y Combinator

Do the following exercises from the Watt text:

(a) Exercise 5.8, page 145.

(b) Exercise 5.9, page 145.

2. Evaluation Order

There is a method to test whether an interpreter for Scheme uses applicative-order evaluation or normal-order evaluation. Define the following two procedures:

```
(define (p) (p))

(define (test x y)
  (if (= x 0)
      0
      y
  )
)
```

Suppose you evaluate the expression `(test 0 (p))`.

(a) What behavior will you observe with an interpreter that uses *applicative-order* evaluation? Explain.

(b) What behavior will you observe with an interpreter that uses *normal-order* evaluation? Explain.

(c) What evaluation order does Scheme use?

Assume that the evaluation rule for the ‘‘special’’ form

(`if` *predicate-expression then-expression else-expression*)

is the same whatever evaluation order is used: the predicate-expression is evaluated *first*, and that result determines whether to evaluate the then-expression or the else- expression. Only one or the other of these two expressions is ever evaluated.

3. Normal Form

Define $\mathbf{S} = \lambda x . \lambda y . \lambda z . xz(yz)$ and $\mathbf{K} = \lambda u . \lambda v . u$.

(a) Give a diagram showing all β-reduction sequences to normal form of

$$\mathbf{SKK} = ((\lambda x . \lambda y . \lambda z . xz(yz))(\lambda u . \lambda v . u))(\lambda u . \lambda v . u)$$

(b) Highlight the call-by-name (outermost) and call-by-value (innermost) reduction sequences in the diagram.

(c) You know that the "self-apply" expression $(\lambda x . xx)$ cannot be given a consistent type and so cannot be defined in the typed lambda calculus. But **S** and **K** can defined in a typed λ-calculus. What are their signatures? Be as general as possible. Interpret your findings in part (a) as an identity in the typed λ-calculus, and describe what it says.

4. Typing and ML

The *composition* functional $\mathbf{B} = \lambda fgx . f(g(x))$ can be defined as $\mathbf{B} = \mathbf{S(KS)K}$.

(a) Using the definition in terms of **S** and **K**, show that **B** has the desired properties by proving via reductions that

$$\mathbf{B}xyz = x(yz)$$

(b) Build **B** from **S** and **K** in Standard ML, and show thereby that it has a consistent type. Give the type. Show via testing that your resulting functional has the composition property $\mathbf{B}fgx = f(g(x))$.

(c) In lambda calculus one can define the combinator **C** via

$$\mathbf{C} = \mathbf{S(BBS)(KK)}$$

Show by reduction that **C** has the property

$$\mathbf{C}xyz = xzy \ .$$

(d) Is **C** type consistent? If so, use Standard ML to construct **C** and its polymorphic type. If not, prove that it is type-inconsistent, using the rules of typed λ-calculus.

5. A Function Domain

(a) Diagram the partial order of all monotone functions from **Truth – Value** to **Truth – Value**. That is, give a complete description of the functional domain (**Truth – Value** → **Truth – Value**). Represent each function by a little diagram, as in class, showing which of $\{true, \ false, \ \bot\}$ is mapped to which of $\{true, \ false, \ \bot\}$.

(b) Exhibit a non-monotonic function from $\{true, \ false, \ \bot\}$ to $\{true, \ false, \ \bot\}$, and explain why it is impossible to implement this as a logic gate, where ⊥ means "signal not yet received", *false* means "signal negative voltage" and *true* means "signal positive voltage."

(c) Consider the (more complicated) domain (**Truth – Value** × **Truth – Value** → **Truth – Value**). Among all the functions in this domain, indicate by a truth table or diagram which function corresponds to each of the following:

(i) Ordinary **and**

(ii) Ordinary **or**

(iii) Short-circuit **and** (sometimes called "conditional and" or **cand**). Assume left to right argument evaluation.

(iv) Short-circuit **or** (sometimes called "conditional or" or **cor**) Assume left to right argument evaluation.

(d) The function called "parallel and'' or **parand** behaves like this: **parand**$(\bot, false) = false$, **parand**$(false, \bot) = false$, **parand**$(false, true) = false$, **parand**$(true, false) = false$, **parand**$(true, true) = true$, **parand**$(false, false) = false$, with all other cases evaluating to ⊥.

Is **parand** a monotone function? Why or why not?