Version of December 9, 1997, 10:18pm

MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
9 December 1997

The Halting Problem

1 A Self-Reproducing Expression

As a warmup for the logic, let `<make-self-app>` be an expression such that

```
(<make-self-app> '<expr>) ;input expression
```

evaluates to

```
(<expr> '<expr>)          ;output notation
```

Namely,

```
(define (make-self-app quoted-expr)
  (list quoted-expr
        (list 'quote quoted-expr)))
```

Then evaluate

```
(make-self-app 'make-self-app)
```

and get the output

```
;Value: (make-self-app 'make-self-app)
```

SELF-REPRODUCTION!?!

2 The Scheme Halting Problem

A Scheme expression is said to HALT if it returns a value when it is evaluated (exactly once, in the Scheme initial environment).

Is there a single Scheme expression, `<halts?>`, which decides whether an expression halts? Namely, can it be that for *every* Scheme expression, `<expr>`, we have

CONTRACT:

`(<halts?> '<expr>)` returns `#f`
IFF `<expr>` does NOT halt.

NOTE: Like an intepreter, `<halts?>` gets the *quoted* `<expr>` as its argument, and so it can "study" `<expr>` without necessarily evaluating it.

3 Proof by Contradiction

So *suppose* there was a `<halts?>`.

Let `<perverse>` be the lambda expression:

```
(lambda (expr)
  (if (<halts?>
       (list expr (list 'quote expr)))
      (/ 1 0)    ;ERROR DOES NOT "HALT"
      #t))
```

Then for EVERY expression `<expr>`, we have by definition
`(<expr> '<expr>)` does *not* halt iff
`(<halts?> '(<expr> '<expr>))` returns `#f`
iff `(<perverse> '<expr>)` returns `#t`
iff `(<perverse> '<expr>)` *does* halt.

Summary: for every expression `<expr>`,
`(<expr> '<expr>)` does *not* halt iff
`(<perverse> '<expr>)` *does* halt.

In particular, choose `<expr>` be the expression `<perverse>`, so

`(<perverse> '<perverse>)` doesn't halt
iff it does halt.

CONTRADICTION!