Exercise 4.15.

Given a one-argument procedure `p` and an object `a`, `p` is said to “halt” on `a` if evaluating the expression `(p a)` returns a value (as opposed to terminating with an error message or running forever). Show that it is impossible to write a procedure `halts?` that correctly determines whether `p` halts on `a` for any procedure `p` and object `a`. Use the following reasoning: If you had such a procedure `halts?`, you could implement the following program:

```
(define (run-forever) (run-forever))

(define (try p)
  (if (halts? p p)
      (run-forever)
      ’halted))
```

Now consider evaluating the expression `(try try)` and show that any possible outcome (either halting or running forever) violates the intended behavior of `halts?`.[1]

Answer.

The procedure `halts?` in our hands is supposed to tell whether a given procedure `p` halts on an object `a`. The begining of such a procedure may looks like:

```
(define halts?
  (lambda (p a)
    ...))
```

It is obvious that `halts?` either returns `#f` or `#t`, depending on whether `p` stops when applied to `a`. By evaluating the expression `(try try)`, we are exploring whether `try` stops and returns a value when applied to itself. The process generated by the evaluation is:

```
(try try)
(if (halts? try try)
    (run-forever)
    ’halted)
```

Since there are only two possible values for the predicate `(halts? try try)`. Let’s say that it is `#f`, which really means that `try` will not stop on itself. But according to the process `(try try)` generated above, `try` is supposed to halt on itself and return `’halted` when `(halts? try try)` evaluates to `#f`. This violates the intended behavior of `halts?`.

Obviously, we must have been wrong about `(halts? try try)`. It must returns `#t`, because `halts?` either returns `#f` or `#t`. This indicates that `try` will halt on itself. But the process `(try try)` generated above shows that `try` will run forever when `(halts? try try)` evaluates to `#t`. This again involves us into contradiction.

So far, all our reasonable deduction shows that any possible outcome (either halting or running forever) violates the intended behavior of `halts?`. Therefore it is impossible to write a procedure `halts?` that correctly determines whether `p` halts on `a` for any procedure `p` and object `a`.

[1] Although we stipulated that `halts?` is given a procedure object, notice that this reasoning still applies even if `halts?` can gain access to the procedure’s text and its environment. This is Turing’s celebrated *Halting Theorem*, which gave the first clear example of a *non-computable* problem, i.e., a well-posed task that cannot be carried out as a computational procedure.