The Halting Problem

**Input:** a specification of a procedure $P$

**Output:** If evaluating an application of $P$ halts, output *true*. Otherwise, output *false*.

---

**Halting Examples**

```scheme
> (halts? '(lambda () (+ 3 3)))
#t
> (halts? '(lambda () (define (f) (f)) (f)))
#f
> (halts? `(lambda ()
           (define (fibo n)
             (if (or (= n 1) (= n 2)))
             (+ (fibo (- n 1)) (fibo (- n 2)))))
           (fibo 100))
#t
```

---

**Can we define halts? ?**

- We could try for a really long time, get something to work for simple examples, but could we solve the problem – make it work for all possible inputs?

---

**Informal Proof**

```scheme
(define (paradox)
  (if (halts? 'paradox)
      (loop-forever)
      #t))
```

If paradox halts, the if test is true and it evaluates to (loop-forever) - it doesn’t halt!

If paradox doesn’t halt, the if test if false, and it evaluates to #t. It halts!
Proof by Contradiction
1. Show \( X \) is nonsensical.
2. Show that if you have \( A \) you can make \( X \).
3. Therefore, \( A \) must not exist.

\[ X = \text{paradox} \]
\[ A = \text{halts? algorithm} \]

How convincing is our Halting Problem proof?
\[
\text{(define (paradox)}
\text{(if (halts? 'paradox)}
\text{(loop-forever)}
\text{#t)})
\]
If contradict-halts halts, the if test is true and it evaluates to (loop-forever) - it doesn’t halt!
If contradict-halts doesn’t halt, the if test is false, and it evaluates to #t. It halts!

This “proof” assumes Scheme exists and is consistent! Scheme is too complex to believe this...we need a simpler model of computation (in two weeks).

“Evaluates to 3” Problem
Input: A procedure specification \( P \)
Output: \text{true} if evaluating \((P)\) would result in 3; \text{false} otherwise.

Is “Evaluates to 3” computable?

Proof by Contradiction
1. Show \( X \) is nonsensical.
2. Show that if you have \( A \) you can make \( X \).
3. Therefore, \( A \) must not exist.

\[ X = \text{halts? algorithm} \]
\[ A = \text{evaluates-to-3? algorithm} \]

Undecidability Proof
Suppose we could define evaluates-to-3? that decides it. Then we could define halts?:
\[
\text{(define (halts? P)}
\text{(evaluates-to-3?}
\text{ '(lambda () (begin (P) 3))))}
\]
if #t: it evaluates to 3, so we know \((P)\) must halt.
if #f: the only way it could not evaluate to 3, is if \((P)\) doesn’t halt. (Note: assumes \((P)\) cannot produce an error.)

Hello-World Problem
Input: An expression specification \( E \)
Output: \text{true} if evaluating \( E \) would print out “Hello World!”; \text{false} otherwise.

Is the Hello-World Problem computable?
Uncomputability Proof
Suppose we could define \texttt{prints-hello-world?} that solves it. Then we could define \texttt{halts?:}

```scheme
(define (halts? P)
  (prints-hello-world?
   'begin ((remove-prints P))
   (print "Hello World!")))
```

Proof by Contradiction
1. Show \( X \) is nonsensical.
2. Show that if you have \( A \) you can make \( X \).
3. Therefore, \( A \) must not exist.

\[ X = \text{halts? algorithm} \]
\[ A = \text{prints-hello-world? algorithm} \]

Charge
- Next week:
  - Monday: computability of virus detection, AI/programming problem; history of Object-Oriented programming
  - Wednesday, Friday: implementing interpreters
- After next week:
  - Scheme is very complicated (requires more than 1 page to define)
  - To have a convincing proof, we need a simpler programming model in which we can write paradox: Turing's model