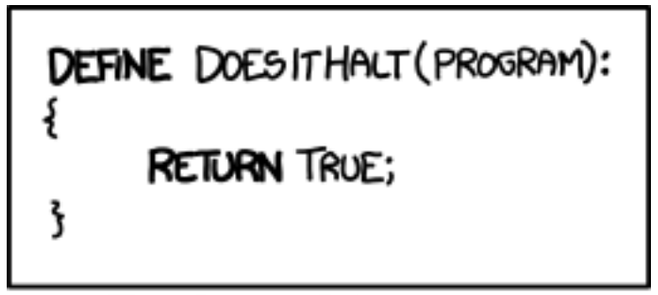
The Halting Problem

```
((lambda (x) (x x)) (lambda (x) (x x)))
```

Is it possible to write such a program?

```
(define (halts? program arg)
  ; Returns true if (program arg) halts,
  ; false if it doesn't
  )
```

http://xkcd.com/1266



THE BIG PICTURE SOLUTION
TO THE HALTING PROBLEM

This statement is false.

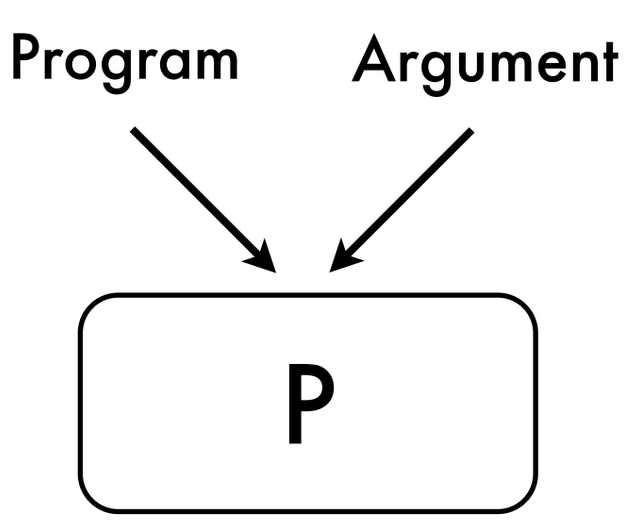
Scooping the Loop Snooper

An elementary proof on the undecidability of the halting problem

Geoffrey K. Pullum, University of Edinburgh

No general procedure for bug checks will do.
Now, I won't just assert that, I'll prove it to you.
I will prove that although you might work till you drop,
you cannot tell if computation will stop.

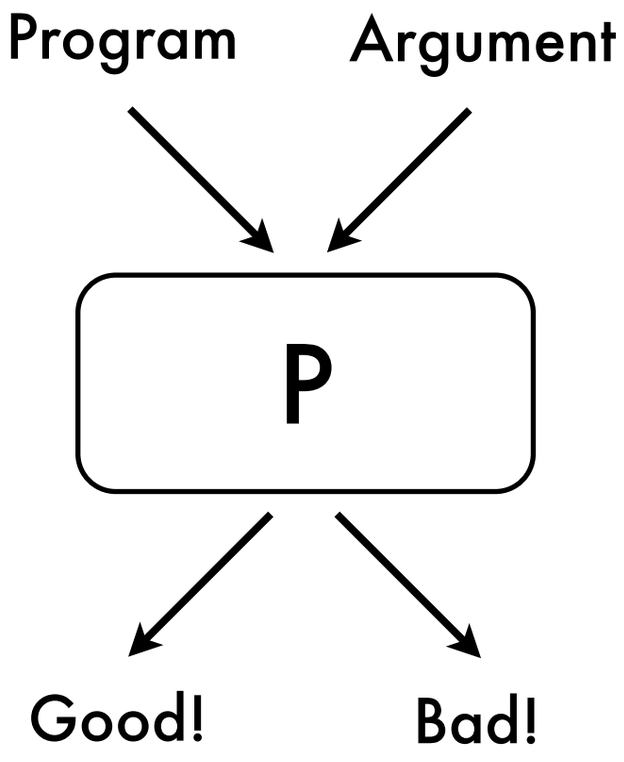
For imagine we have a procedure called P
that for specified input permits you to see
whether specified source code, with all of its faults,
defines a routine that eventually halts.




```
(define (P program arg)
  ; Returns true if (program arg) halts,
  ; false if it doesn't
  )
```

You feed in your program, with suitable data,
and P gets to work, and a little while later
(in finite compute time) correctly infers
whether infinite looping behavior occurs.

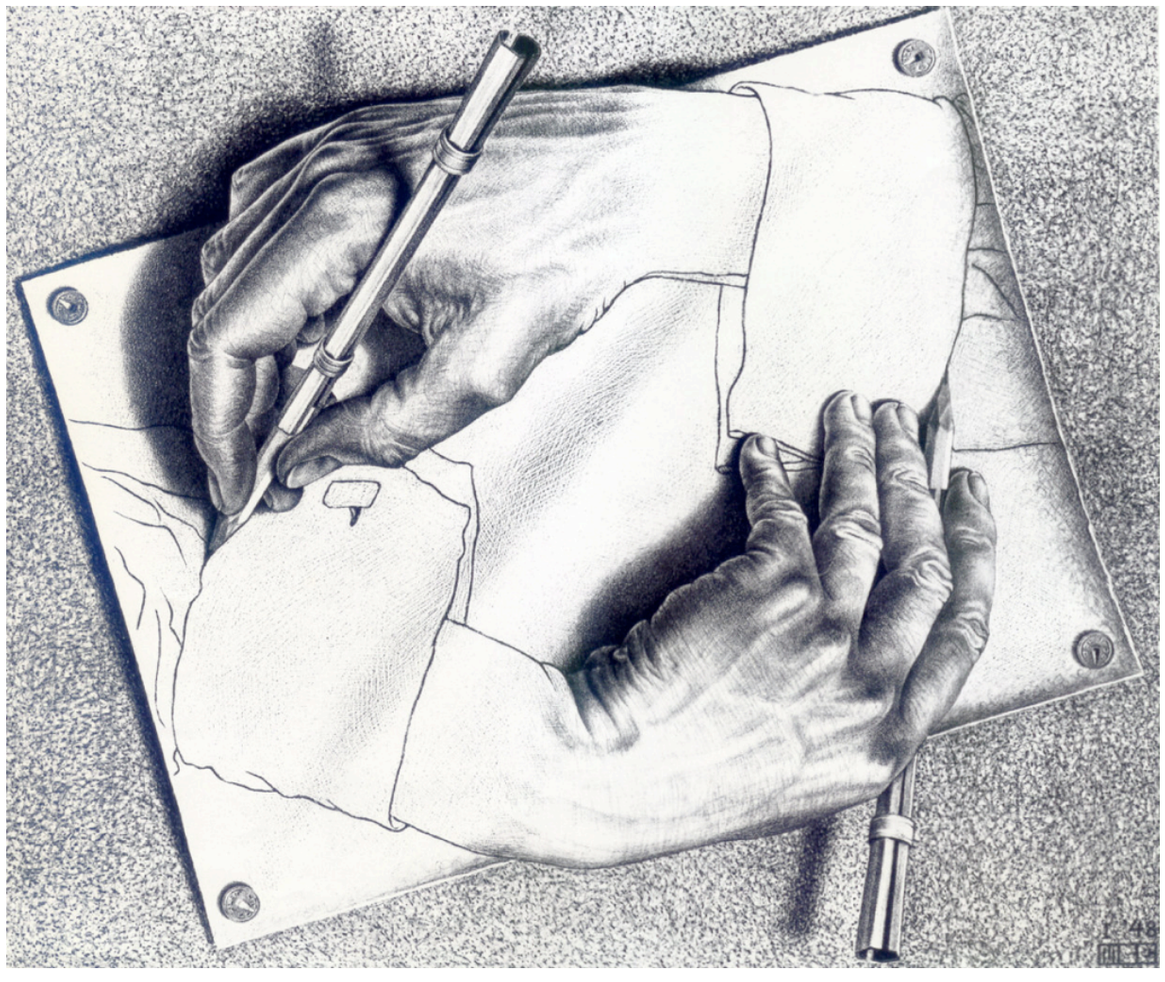
If there will be no looping, then P prints out ‘Good.’
That means work on this input will halt, as it should.
But if it detects an unstoppable loop,
then P reports ‘Bad!’ — which means you’re in the soup.




```
(P '(lambda (x) (* x x) 4))
; Returns true, as (* 4 4) halts

(P '(lambda (x) (x x)) '(lambda (x) (x x)))
; Returns false, as above program
; loops infinitely
```

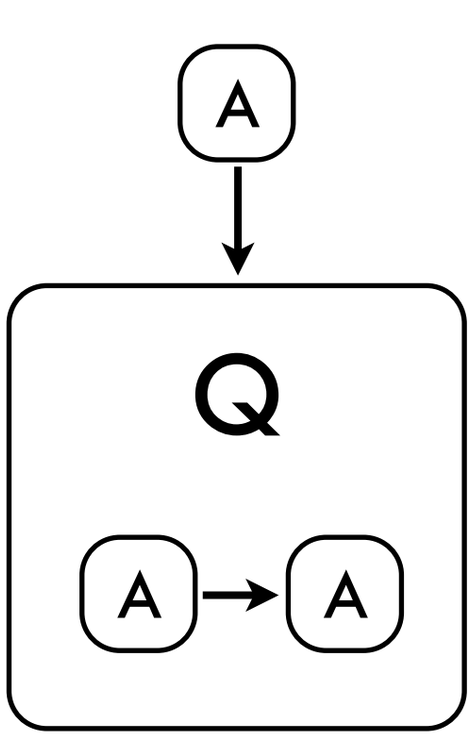
Well, the truth is that P cannot possibly be,
because if you wrote it and gave it to me,
I could use it to set up a logical bind
that would shatter your reason and scramble your mind.



Drawing Hands, by M.C. Escher

Here's the trick that I'll use — and it's simple to do.
I'll define a procedure, which I will call Q,
that will use P's predictions of halting success
to stir up a terrible logical mess.

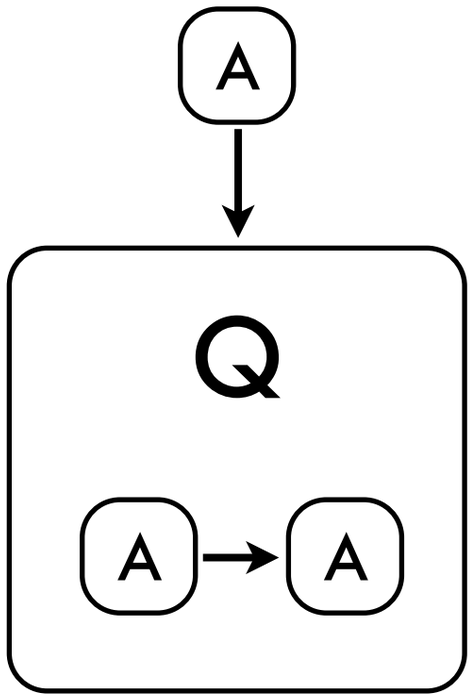
For a specified program, say A, one supplies,
the first step of this program called Q I devise
is to find out from P what's the right thing to say
of the looping behavior of A run on A.




```
(define A
  (lambda (x) (cdr x)))

(A '(lambda (x) (cdr x)))
((x) (cdr x))
```

If P's answer is 'Bad!', Q will suddenly stop.
But otherwise, Q will go back to the top,
and start off again, looping endlessly back,
till the universe dies and turns frozen and black.




```
(define Q
  '(lambda (A)
     (if (P A 'A)
         ; Infinitely loops if A halts on itself
         ((lambda (x) (x x)) (lambda (x) (x x)))
         ; Otherwise halt
         #t)))
```

And this program called Q wouldn't stay on the shelf;
I would ask it to forecast its run on itself.
When it reads its own source code, just what will it do?
What's the looping behavior of Q run on Q?

If P warns of infinite loops, Q will quit;
yet P is supposed to speak truly of it!
And if Q's going to quit, then P should say 'Good.'
Which makes Q start to loop! (P denied that it would.)

```
(define Q
  '(lambda (A)
     (if (P A 'A)
         ; Infinitely loops if A halts on itself
         ((lambda (x) (x x)) (lambda (x) (x x)))
         ; Otherwise halt
         #t)))

(P Q Q)
```

No matter how P might perform, Q will scoop it:
Q uses P's output to make P look stupid.
Whatever P says, it cannot predict Q:
P is right when it's wrong, and is false when it's true!

I've created a paradox, neat as can be –
and simply by using your putative P.
When you posited P you stepped into a snare;
Your assumption has led you right into my lair.

So where can this argument possibly go?
I don't have to tell you; I'm sure you must know.
A reductio: There cannot possibly be
a procedure that acts like the mythical P.

You can never find general mechanical means
for predicting the acts of computing machines;
it's something that cannot be done. So we users
must find our own bugs. Our computers are losers!

Implications

- The `halts?` function is uncomputable
- We cannot, in general, tell if two programs compute to the same result
- Perfect anti-virus software is theoretically impossible
- Can we have a computer to do mathematics for us?

Implications

Goldbach's conjecture:

Every even integer greater than 2 can be expressed as the sum of two primes

```
(define (find-goldbach n)
  ; halts if goldbach's conjecture is false
  (define (goldbach n)
    ; Tries every prime up to n
    ; Returns true if n is the sum of two primes
    )
  (if (goldbach n)
      (find-goldbach (+ n 2))
      #f))

(halts? find-goldbach 4)
```

Gödel's Incompleteness Theorem

In any formal system for arithmetic, there exists true mathematical statements that are unprovable