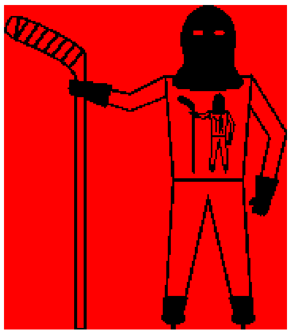
Class 24: Computability



Halting Problems Hockey Team Logo

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

Menu

- Review:
  - Gödel's Theorem
  - Proof in Axiomatic Systems
- Computability:

  Are there some problems that it is impossible to write a program to solve?

Gödel's Proof

G: This statement of number theory does not have any proof in the system of *PM*.

If *G* were provable, then PM would be inconsistent.

If *G* is unprovable, then PM would be incomplete.

PM cannot be complete and consistent!

What does it mean for an axiomatic system to be complete and consistent?

Derives **all** true statements, and **no** false statements starting from a finite number of axioms and following mechanical inference rules.

What does it mean for an axiomatic system to be complete and consistent?

It means the axiomatic system is weak.

Its is so weak, it cannot express "This statement has no proof."

Why is an *Inconsistent* Axiomatic System less useful than an *Incomplete* Axiomatic System?

Inconsistent Axiomatic System

Derives **all** true statements, and **some** false statements starting from a finite number of axioms and following mechanical inference rules.

some false statements

Once you can prove one false statement, everything can be proven! false $\Rightarrow$ anything

Proof

- A proof of S in an axiomatic system is a sequence of strings, T_0, T_1, …, T_n where:
  - The first string is the axioms
  - For all i from 1 to n, T_n is the result of applying one of the inference rules to T_{n-1}
  - T_n is S
- How much work is it to **check** a proof?

Proof Checking Problem

- Input: an axiomatic system (a set of axioms and inference rules), a statement S, and a proof P containing n steps of S
- Output:

  true if P is a valid proof of S

  false otherwise

How much work is a proof-checking procedure?

We can write a proof-checking procedure that is $\theta(n)$

Finite-Length Proof Finding Problem

- Input: an axiomatic system (a set of axioms and inference rules), a statement S, n (the maximum number of proof steps)
- Output: A valid proof of S with no more then n steps if there is one. If there is no proof of S with <= n steps, **unprovable.**

How much work?

At worst, we can try all possible proofs:
r inference rules, 0 - n steps ~ r^n possible proofs
Checking each proof is $\theta(n)$
So, there is a procedure that is $\theta(nr^n)$
but, it might not be the best one.

Proof Finding Problem

- Input: an axiomatic system, a statement S
- Output: If S is true, output a valid proof. If S is not true, output **false**.

How much work?

It is **impossible**!

"It might take infinite work."

Gödel's theorem says it cannot be done.

Computability

Algorithms

- What's an algorithm?

  A procedure that always terminates.
- What's a procedure?

  A precise (mechanizable) description of a process.

Computability

- Is there an algorithm that solves a problem?
- Decidable (computable) problems:
  - There is an algorithm that solves the problem.
  - Make a photomosaic, sorting, drug discovery, winning chess (it doesn't mean we know the algorithm, but there is one)
- Undecidable problems:
  - There is no algorithm that solves the problem.

    There might be a procedure, but it doesn't always terminate.

Are there any undecidable problems?

The Proof-Finding Problem:

- Input: an axiomatic system, a statement S
- Output: If S is true, output a valid proof. If S is not true, output **false**.

Any others?

How would you prove a problem is undecidable?

Hint: how did we prove 3-SAT was NP-Complete (once we knew Smiley Puzzle was?

Undecidable Problems

- We can prove a problem is undecidable by showing it is at least as hard as the proof-finding problem
- Here's a famous one:

  Halting Problem

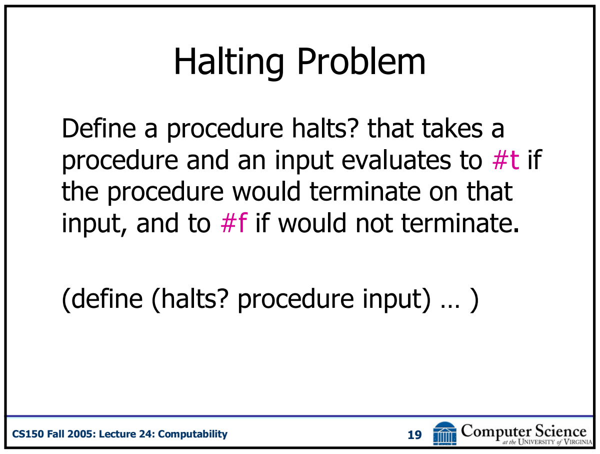
  Input: a procedure P (described by a Scheme program) and its input I

  Output: true if executing P on I halts (finishes execution), false otherwise.

Alan Turing (1912-1954)

- Codebreaker at Bletchley Park
  - Broke Enigma Cipher
  - Perhaps more important than Lorenz
- Published *On Computable Numbers …* (1936)
  - Introduced the Halting Problem
  - Formal model of computation (now known as "Turing Machine")
- After the war: convicted of homosexuality (then a crime in Britain), committed suicide eating cyanide apple

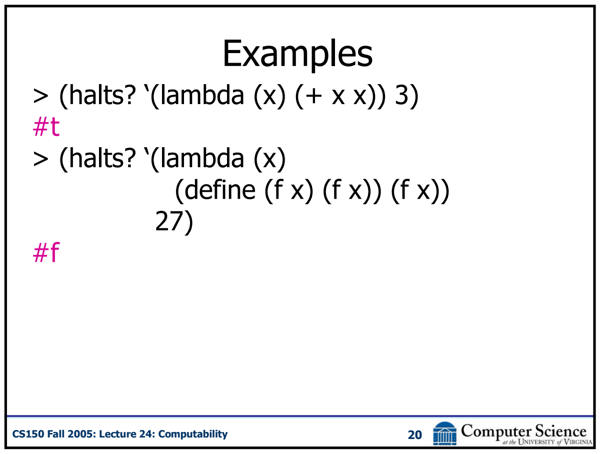
5 years after Gödel's proof!



Halting Problem

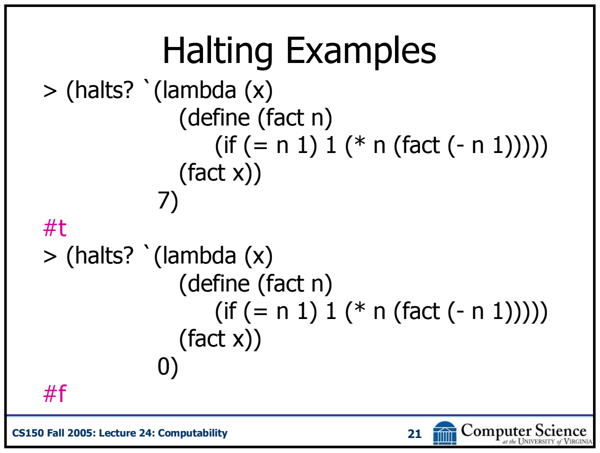
Define a procedure halts? that takes a procedure and an input evaluates to #t if the procedure would terminate on that input, and to #f if would not terminate.

```
(define (halts? procedure input) … )
```



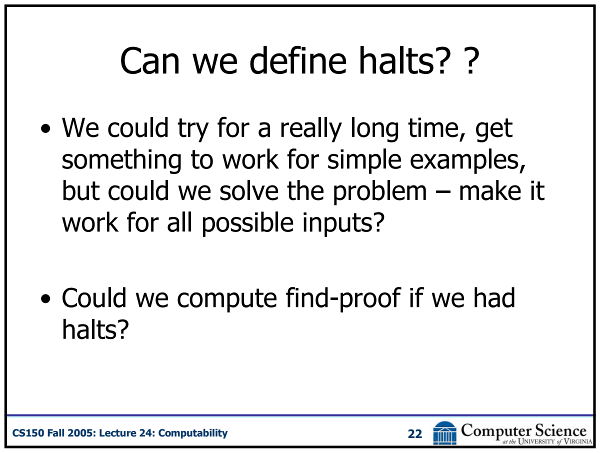
Examples

```
> (halts? '(lambda (x) (+ x x)) 3)
#t
> (halts? '(lambda (x)
              (define (f x) (f x)) (f x))
           27)
#f
```



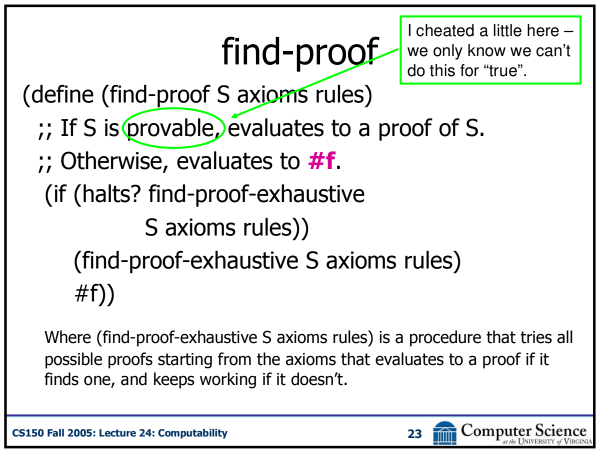
Halting Examples

```
> (halts? `(lambda (x)
              (define (fact n)
                 (if (= n 1) 1 (* n (fact (- n 1)))))
              (fact x))
           7)
#t
> (halts? `(lambda (x)
              (define (fact n)
                 (if (= n 1) 1 (* n (fact (- n 1)))))
              (fact x))
           0)
#f
```



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?
- Could we compute find-proof if we had halts?

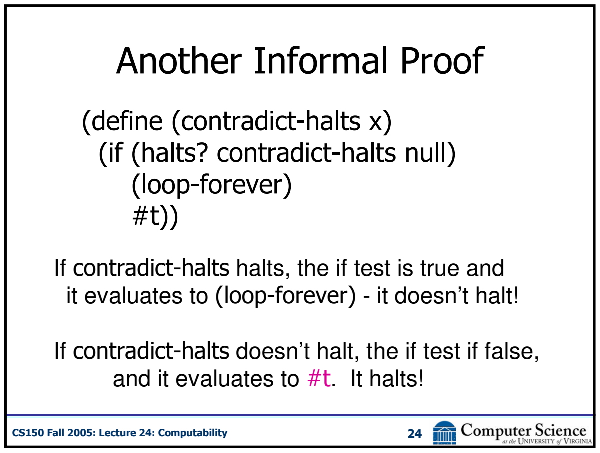


find-proof

I cheated a little here – we only know we can't do this for "true".

```
(define (find-proof S axioms rules)
  ;; If S is provable, evaluates to a proof of S.
  ;; Otherwise, evaluates to #f.
   (if (halts? find-proof-exhaustive
               S axioms rules))
      (find-proof-exhaustive S axioms rules)
      #f))
```

Where (find-proof-exhaustive S axioms rules) is a procedure that tries all possible proofs starting from the axioms that evaluates to a proof if it finds one, and keeps working if it doesn't.



Another Informal Proof

```
(define (contradict-halts x)
  (if (halts? contradict-halts null)
      (loop-forever)
      #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 if false, and it evaluates to #t. It halts!

Reducing Undecidable Problems

- If solving a problem *P* would allow us to solve the halting problem, then *P* is undecidable – there is no solution to *P*, since we have proved there is no solution to the halting problem!
- There are **lots of** important problems like this
  - Friday: why virus scanners will never work perfectly

Charge

- Now (if you can)
  - Marc Levoy, Newcomb South Meeting Room "Digital Michelangelo"