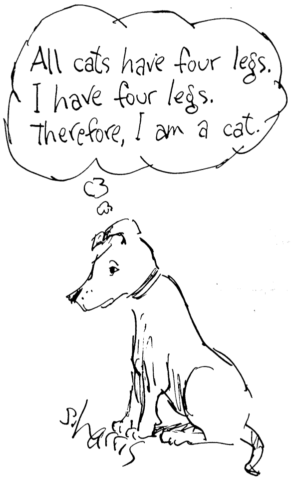
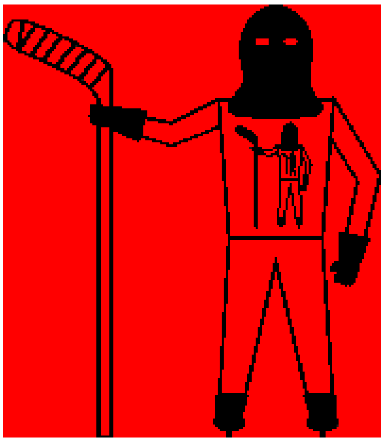
Godel and Computability





Halting Problems Hockey Team

One-Slide Summary

- A **proof** of X in a formal system is a sequence of steps starting with axioms. Each step must use a valid rule of inference and the final step must be X.
- All interesting logical systems are **incomplete**: there are true statements that cannot be proven within the system.
- An **algorithm** is a (mechanizable) procedure that always terminates.
- A problem is **decidable** if there exists an algorithm to solve it. A problem is **undecidable** if it is not possible for an algorithm to exists that solves it.
- The **halting problem** is undecidable.

Outline

- Gödel's Proof
- Unprovability
- Algorithms
- Computability
- The Halting Problem

Epimenides Paradox

Epimenides (a Cretan):
“All Cretans are liars.”

Equivalently:
“This statement is false.”

Russell’s types can help with the set paradox, but not with these.

Gödel’s Solution

All consistent axiomatic formulations of number theory include *undecidable* propositions.

(GEB, p. 17)

undecidable – cannot be proven either true or false inside the system.

Kurt Gödel

- Born 1906 in Brno (now Czech Republic, then Austria-Hungary)
- 1931: publishes *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme* (*On Formally Undecidable Propositions of Principia Mathematica and Related Systems*)

- 1939: flees Vienna
- Institute for Advanced Study, Princeton
- Died in 1978 – convinced everything was poisoned and refused to eat

Gödel's Theorem

In the Principia Mathematica system, there are statements that cannot be proven either true or false.

Gödel's Theorem

In any interesting rigid system, there are statements that cannot be proven either true or false.

Gödel's Theorem

All logical systems of any complexity are **incomplete**: there are statements that are *true* that cannot be proven within the system.

Proof – General Idea

- Theorem: In the Principia Mathematica system, there are statements that cannot be proven either true or false.
- Proof: Find such a statement!

Gödel's Statement

G: This statement does not have any proof in the system of *Principia Mathematica*.

G is unprovable, but true!

Why?

Gödel's Statement

G: This statement does not have any proof in the system.

Possibilities:

1. *G* is **true** ⇒ *G* **has no** proof
   System is *incomplete*
2. *G* is **false** ⇒ *G* **has a** proof
   System is *inconsistent*

Gödel's Proof Idea

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

If *G* is provable, PM would be inconsistent.
If *G* is unprovable, PM would be incomplete.

Thus, **PM cannot be complete and consistent!**

Finishing The Proof

- Turn *G* into a statement in the *Principia Mathematica* system
- Is *PM* powerful enough to express "This statement does not have any proof in the *PM* system."?

How to express "does not have any proof in the system of *PM*"

- What does "**have a proof** of *S* in PM" mean?
  - There is a sequence of steps that follow the inference rules that starts with the initial axioms and ends with *S*
- What does it mean to "**not** have **any** proof of *S* in PM"?
  - There is **no** sequence of steps that follow the inference rules that starts with the initial axioms and ends with *S*

Can PM express unprovability?

- There is **no** sequence of steps that follows the inference rules that starts with the initial axioms and ends with S
- Sequence of steps:

  $T_0, T_1, T_2, \ldots, T_N$

T_0 must be the axioms
T_N must include S
Every step must follow from the previous using an inference rule

Can we express "This statement"?

- Yes!
  - Optional Reading: the TNT Chapter in GEB
- We can write turn every statement into a number, so we can turn "This statement does not have any proof in the system" into a number

Gödel's Proof

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

If *G* is provable, PM would be inconsistent.

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

PM can express *G*.

Thus, **PM cannot be complete and consistent!**

Generalization

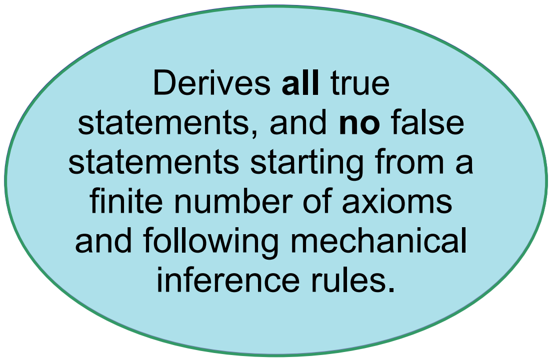
All logical systems of any complexity are incomplete:

there are statements that are *true* that cannot be proven within the system.

Practical Implications

- Mathematicians will ***never*** be completely replaced by computers
  - There are mathematical truths that cannot be determined mechanically
  - We can build a computer that will prove only true theorems about number theory, but if it cannot prove something we do not know that that is not a true theorem.

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

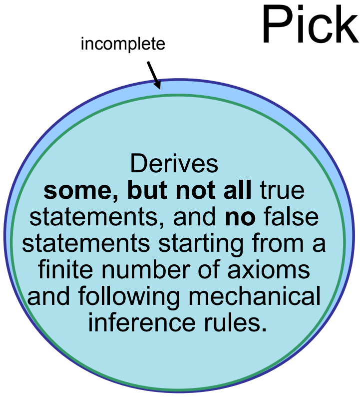


What does it mean for an axiomatic system to be complete and consistent?

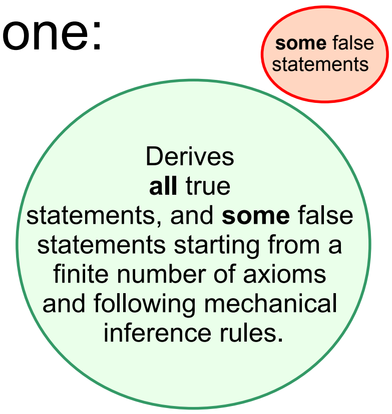
It means the axiomatic system is weak.

Indeed, it is *so* weak, it cannot express: "This statement has no proof."

Pick one:



Incomplete Axiomatic System



Inconsistent 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 ⇒ anything

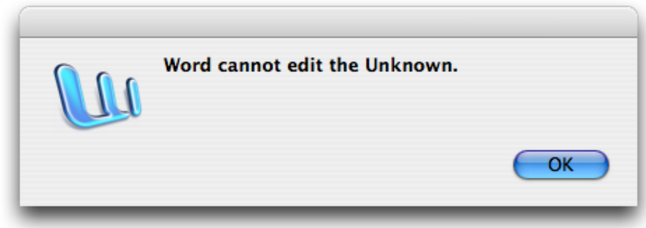
Algorithms

- 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?
- **Computable** (**decidable**) 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)
- **Uncomputable** (**undecidable**) problems:
  - There is no algorithm that solves the problem.
  - There might be a procedure, but it doesn't always terminate.

Are there any uncomputable problems?



The Halting Problem

Input: a specification of a procedure P

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

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 specification and evaluates to #t if evaluating an application of the procedure would terminate, and to #f if evaluating an application of the would not terminate.

```
(define (halts? proc) … )
```


Examples

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

Halting Examples

```
> (halts? `(lambda ()
             (define (fact n)
                (if (= n 1) 1 (* n (fact (- n 1)))))
             (fact 7)))
#t
> (halts? `(lambda () (fact 0)))
#f
> (halts? `(lambda ()
             (define (fibo n)
                (if (or (= n 1) (- n 2))) 1
                   (+ (fibo (- n 1)) (fibo (- n 2))))))
             (fibo 100))
#t
```

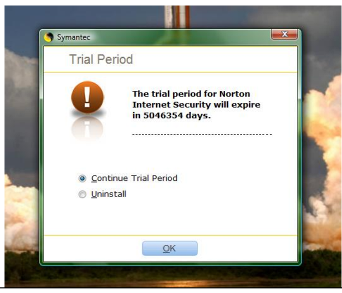

Halting Examples

```
> (halts? `(lambda ()
             (define (sum-of-two-primes? n)
                ;;; try all possibilities... )
             (define (test-goldbach n)
                (if (not (sum-of-two-primes? n))
                    #f ; Goldbach Conjecture wrong
                    (test-goldbach (+ n 2))))
             (test-goldbach 2))
?
```

Goldbach Conjecture (see GEB, p. 394):
Every even integer can be written as the sum of two primes.

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

```
(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

Goal: Show that *A* is false.

1. Show *X* is nonsensical.
2. Show that if you have *A* you can make *X*.
3. Therefore, *A* must not exist.

X = paradox
A = halts? algorithm

How convincing is our Halting Problem proof?

```
(define (paradox)
  (if (halts? 'paradox)
      (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!

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).

Homework

- Read Chapter 12
- Read Obituary
- PS6 Due Monday