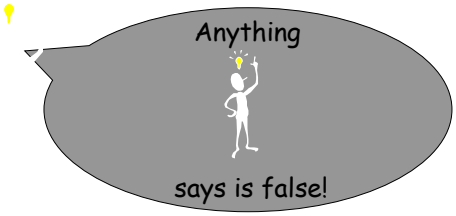
Great Theoretical Ideas In Computer Science

Anupam Gupta		CS 15-251 Fall 2005
Lecture 28	Dec 6, 2005	Carnegie Mellon University

The Limitations of Proofs: An Incompressible Resolution



A Quick Recap of the Previous Lectures

The Halting Problem
K = {P | P(P) halts }

Is there a program HALT such that:

HALT(P) = yes, if P∈K
HALT(P) = no, if P∉K

HALT decides whether or not any given program is in K.

Alan Turing (1912-1954)

Theorem: [1937]

There is no program to solve the halting problem

Computability Theory: Old Vocabulary

We call a set S⊆Σ* decidable or recursive if there is a program P such that:

P(x) = yes, if x∈S
P(x) = no, if x∉S

Hence, the halting set K is undecidable

Computability Theory: Old Vocabulary

We call a set S⊆Σ* enumerable or recursively enumerable (r.e.) if there is a program P such that:

P prints an (infinite) list of strings.
- Any element on the list should be in S.
- Each element in S appears after a finite amount of time.

Enumerating K

```
Enumerate-K {
  for n = 0 to forever {
    for W = all strings of length < n do {
      if W(W) halts in n steps then output W;
    }
  }
}
```

K is not decidable, but it is enumerable!

Let K' = { Java P | P(P) does not halt}

K' is not decidable nor enumerable

Intuitively, a proof is a sequence of "statements", each of which follows "logically" from some of the previous steps.

Super Important Fact

Let S be any (decidable) set of statements.
Let L be any (computable) logic.

We can write a program to enumerate the provable theorems of L.

I.e., $Provable_{S,L}$ is enumerable.

Let S be any decidable language. Let $Truth_S$ be any fixed function from S to True/False.

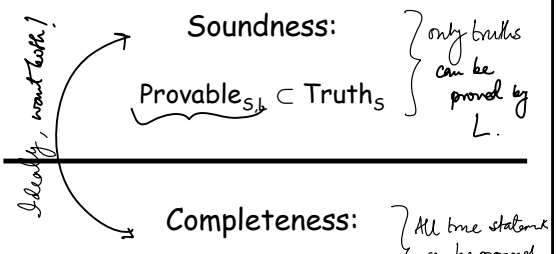
We say $Truth_S$ is a "truth concept" associated with the strings in S.

General Picture:

A decidable set of statements S.

A computable logic L.

A (possibly incomputable) truth concept
$Truth_S$: S → {T, F}



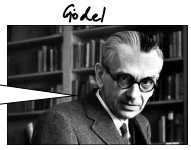
Truth versus Provability

Foundational Crisis:

It is impossible to have a proof system F such that

Provable$_{F,S}$ = Arithmetic_Truth

F is sound for arithmetic will imply F is not complete.



Here's what we have

A language S. (decidable)

A truth concept Truth$_S$. (computable, possibly)

A logic L that is sound (maybe even complete) for the truth concept. (computable)

An enumerable list Provable$_{S,L}$ of provable statements (theorems) in the logic.

Arithmetic_Truth is not enumerable

Suppose Arithmetic_Truth is enumerable, and the program ARITH_LIST enumerates Arithmetic_Truth.

Can now make a program ZERO(polynomial p):

Run ARITH_LIST until either of the two statements appears: "p has an integer root", or "p has no integer root". Output the appropriate answer.

Contradiction of undecidability of D = {set of multivariate polynomials that have at least one integer root}

Hilbert's 10th problem (Lec26)

Arithmetic_Truth has no proof system

There is no sound and complete proof system for Arithmetic_Truth.

Suppose {L,S} is such a proof system. Then there must be a program to enumerate Provable$_{L,S}$.

by completeness & soundness Provable$_{L,S}$ = Truth$_{Arithmetic}$

Provable$_{L,S}$ is r.e.

Arithmetic_Truth is not r.e.

So Provable$_{L,S}$ ≠ Arithmetic_Truth

The existence of integer roots for Diophantine equations was not decidable.

Hence, Arithmetic_Truth is not recursively enumerable.

Hence, Arithmetic_Truth has no sound and complete proof system!!!!

Hilbert's Second Question [1900]

Is there a foundation for mathematics that would, in principle, allow us to decide the truth of any mathematical proposition? Such a foundation would have to give us a clear procedure (algorithm) for making the decision.

Hilbert

Foundation F

Let F be any foundation for mathematics:

1. F is a proof system that only proves true things [Soundness]
2. The set of valid proofs is computable. [There is a program to check any candidate proof in this system]

think of F as (S,L) in the preceding discussion, with L being sound

Gödel's Incompleteness Theorem

In 1931, Kurt Gödel stunned the world by proving that for any consistent axioms F there is a true statement of first order number theory that is not provable or disprovable by F.
I.e., a true statement that can be made using 0, 1, plus, times, for every, there exists, AND, OR, NOT, parentheses, and variables that refer to natural numbers.

Incompleteness

Let us fix F to be any attempt to give a foundation for mathematics. We have already proved that it cannot be sound and complete. Furthermore…

We can even construct a statement that we will all believe to be true, but is not provable in F.

$CONFUSE_F(P)$

Loop though all sequences of sentences in S

If S is a valid F-proof of "P halts",
then loop-forever

If S is a valid F-proof of "P never halts", then halt.

Program $CONFUSE_F(P)$

Loop though all sequences of sentences in S
If S is a valid F-proof of "P halts",
then loop-forever
If S is a valid F-proof of "P never halts", then halt.

Define:

$GODEL_F$ = AUTO_CANNIBAL_MAKER($CONFUSE_F$)

Thus, when we run $GODEL_F$ it will do the same thing as:
$CONFUSE_F(GODEL_F)$

Program CONFUSE$_F$(P)

Loop though all sequences of sentences in S
If S is a valid F-proof of "P halts",
then loop-forever
If S is a valid F-proof of "P never
halts", then halt.

GODEL$_F$ =
AUTO_CANNIBAL_MAKER(CONFUSE$_F$)

Thus, when we run GODEL$_F$ it will do the same thing as CONFUSE$_F$(GODEL$_F$)

Can F prove GODEL$_F$ halts?

If Yes, then CONFUSE$_F$(GODEL$_F$) does not halt
Contradiction (if soundness of F)

Can F prove GODEL$_F$ does not halt?

Yes -> CONFUSE$_F$(GODEL$_F$) halts
Contradiction

F cannot prove or disprove that GODEL$_F$ halts.

GODEL$_F$

F can't prove or disprove that GODEL$_F$ halts. is this statement T or F?

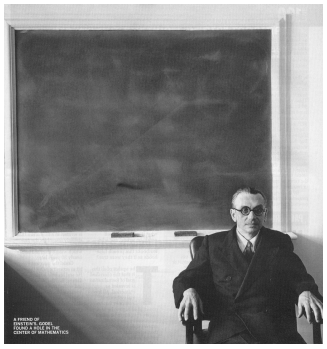
GODEL$_F$ = CONFUSE$_F$(GODEL$_F$) is the following program

Loop though all sequences of sentences in S

If S is a valid F-proof of "P halts",
then loop-forever

If S is a valid F-proof of "P never
halts", then halt.

Hence this program does not halt



To summarize this discussion

F can't prove or disprove that GODEL$_F$ halts.

Thus, CONFUSE$_F$(GODEL$_F$) = GODEL$_F$ will not halt.

Thus, we have just proved what F can't.

F can't prove something that we know is true.
It is not a complete foundation for mathematics.

So what is mathematics?

We can still have rigorous, precise axioms that we agree to use in our reasoning (like the Peano Axioms, or axioms for Set Theory). We just can't hope for them to be complete.

Most working mathematicians never hit these points of uncertainty in their work, but it does happen!

Ancient Paradoxes With An Impossible Resolution.


Leibnitz

Chaitin

Gottfried Wilhelm von Leibniz

There is almost no paradox without utility

LIAR PARADOX:

"This statement is false."

The preceding discussions can be viewed as using this paradox to prove incompleteness.

Bertrand Russell

BERRY PARADOX:

"The smallest natural number that can't be named in less than fourteen words."

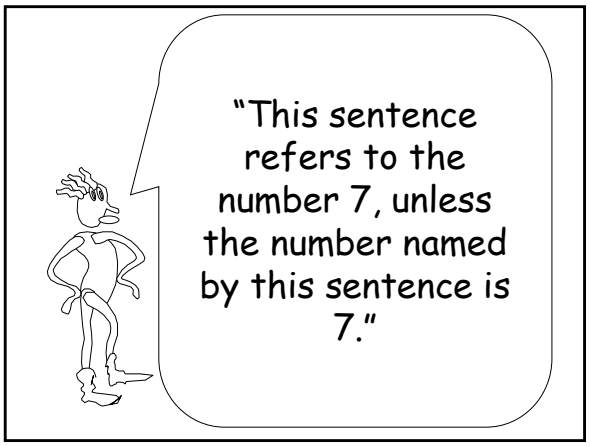
What do we get if we use this paradox instead?

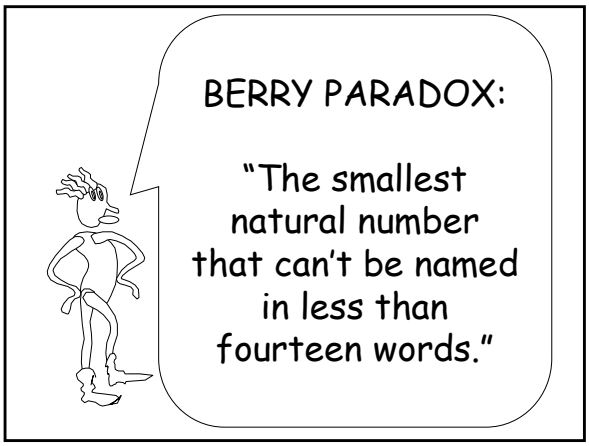
List all English sentences of 13 words or less. For each one, if it names a number, cross that number off a list of natural numbers. Smallest number left is number named by the Berry Sentence?

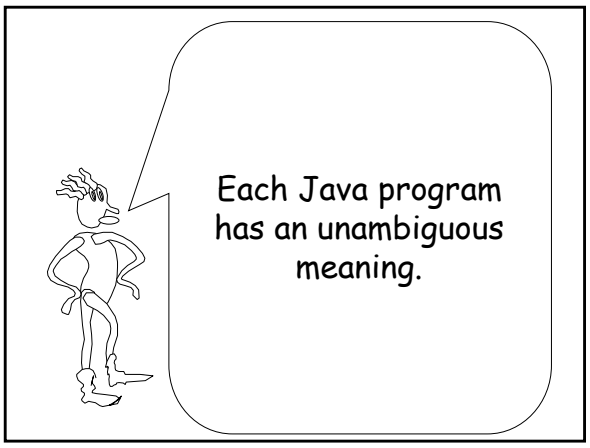
As you loop through sentences, you will meet the Berry sentence. This procedure will not have a well defined outcome.

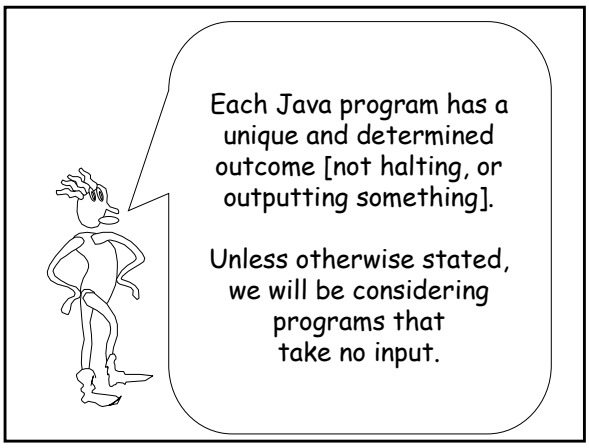
Worse:

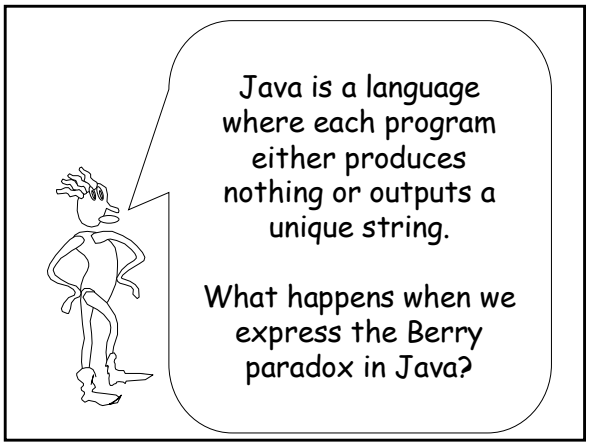
In English, there is not always a fact of the matter about whether or not a given sentence names a number.











Counting

A set of binary stings is "prefix-free" if no string in the set is a prefix of another string in the set

$\{0, 01\}$ is not prefix free

$\{01, 10, 11\}$ is prefix free

Theorem: If S is prefix-free and contains no strings longer than n, then S contains at most 2^n strings.

Proof:

For each string x in S, let f(x) be the string x with n-|X| 0's appended to its right.

Thus, f is a 1-1 map from S into $\{0,1\}^n$.

Hence, $|S|$ must be at most $|\{0,1\}^n| = 2^n$

0 00 10
1 01 11

Binary Java is Prefix-Free

We will represent Java in binary (using ASCII codes for each character). We will allow only java programs where all the classes are put in one big class delimited by { }.

Hence, no JAVA program will be a prefix of any other.

Storing Poker Hands

I want to store a 5 card poker hand using the smallest number of bits (space efficient). The naïve scheme would use 2 bits for a suit, 4 bits for a rank, and hence 6 bits per card and 30 bits per hand. How can I do better?

Order the Poker hands lexicographically

To store a hand all I need is to store its index of size $\lceil \log(2,598,960) \rceil = 22$ bits.

$\lceil \log_2 \binom{52}{5} \rceil$ set S can be represented using $\lceil \log_2 |S| \rceil$

Let's call this the "indexing trick".

22 Bits Is OPTIMAL

$2^{21} < 2,598,560$

There are more poker hands than there are 21 bit strings. Hence, you can't have a string for each hand.

Incompressibility

We call a binary string x incompressible if the shortest Binary Java program to output x is at least as long as x.

Theorem: Half the strings of any given length are incompressible

Theorem: Half the strings of any given length are incompressible

Java is prefix-free so there are at most 2^{n-1} programs of length n-1 or shorter.

There are 2^n strings of length n, and hence at least half of them have no smaller length program that outputs them.

#(of Java programs of length $<n$) $\leq 2^{n-1}$ (because these are prefix-free)

#(of strings of length n) $= 2^n$.

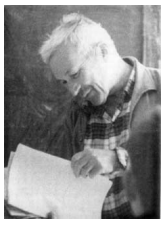
at least $2^n - 2^{n-1}$ of them are incompressible

A compressible string

0101010101010101… a million times …01

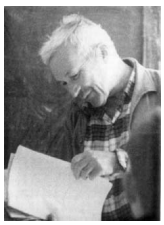
```
public class Counter
{
 public static void main(String argv[])
  {
    for (int i=0; i<1000000; i++)
       System.out.print("01");
  }

}
```

It is possible to *define* randomness in terms of incompressibility



Kolmogorov

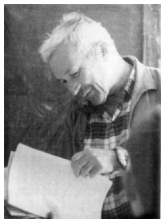
Chaitin



Kolmogorov

Chaitin

An incompressible string has no computable, atypical properties!



Kolmogorov

Chaitin

An incompressible string has no computable pattern!

If a string x is incompressible, then there is nothing atypical that you can say about it.

Suppose D is some atypical, computable predicate that is true of x.

Since D is atypical, it is not true of many n bit strings, where |x| = n.

Now we can compress x by referring to x by its index i in the enumeration of strings of length n that have property D.

[Notice the use of the "indexing trick"]

When we notice a "pattern", we always mean something atypical.

So when you see a "pattern" in a sufficiently long string it allows you to compress it.

Hence, incompressible strings have no pattern.

Now let us get back to the

BERRY PARADOX:

"The smallest natural number that can't be named in less than fourteen words."

Java Berry

The shortest incompressible string that is longer than this Java program

Java Berry

The shortest incompressible string that this program can certify is longer than this program

Incompressibility Detector

Define an Incompressibility Detector to be a program P such that:

P(x) = "yes" means x is definitely incompressible

P(x) = "not sure", otherwise

The program INCOMPRESSIBLE

Let INCOMPRESSIBLE be a JAVA incompressibility detector whose program length is n.

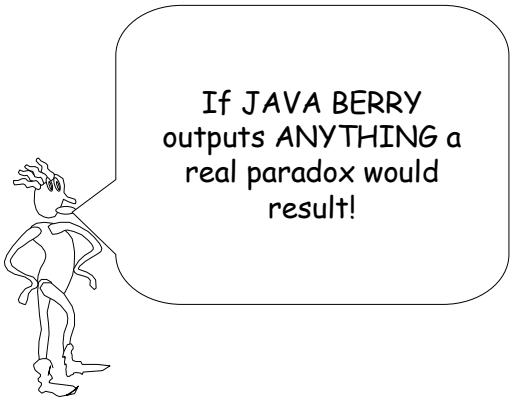
INCOMPRESSIBLE(x) = "yes" means x is definitely incompressible

INCOMPRESSIBLE(x) = "not sure", otherwise

JAVA BERRY

```
{
  k:= bound on length of my program text
  For x = strings of length k+1 to infinity
  {
      If INCOMPRESSIBLE(X) print X;
  }

  Text of subroutine for INCOMPRESSIBLE.
}
```

The shortest incompressible string that this program can certify is longer than this program



JAVA BERRY

```
{
  S = Text of subroutine for INCOMPRESSIBLE;
  n := string-length(S);
  Loop x = strings of length n+b to infinity
  {
    If EXECUTE(S, X) = "YES" print X;
  }

  Routine for EXECUTE(S,X) which executes the
     Java program given by the string S on input X

  Routine for string-length(S) returns the length of string S
}
```

any x of length n+b.

BERRY has text length b + n

Note: b is a constant, independent of n

Java berry outputs nothing

Theorem:

There is a constant b such that no Incompressibility Detector of length n outputs "yes" on any string of length greater than n+b.

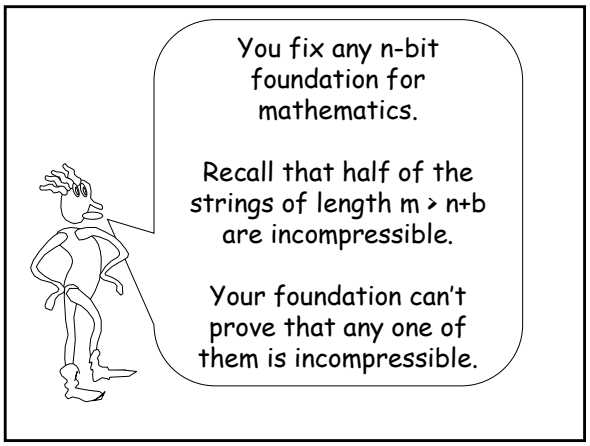
Proof:

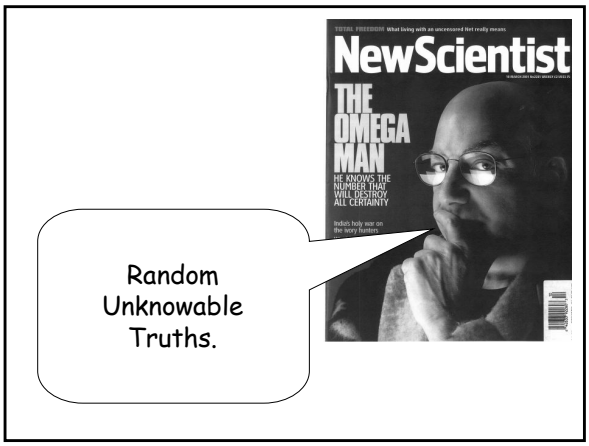
If so, we could use it inside Java Berry and obtain a contradiction.

Half the strings of length n+b are incompressible.

Let Π be a sound, formal system that can be presented as a n-bit program enumerating consequences of its axioms.

No statement of the form
"X is incompressible"
for X of length > n+b
is a consequence of Π.


You fix any n-bit foundation for mathematics.
Recall that half of the strings of length m > n+b are incompressible.
Your foundation can't prove that any one of them is incompressible.


Random Unknowable Truths.
NewScientist
THE OMEGA MAN
HE KNOWS THE NUMBER THAT WILL DESTROY ALL CERTAINTY