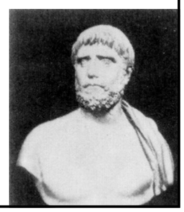
Great Theoretical Ideas In Computer Science		
Steven Rudich		CS 15-251 Spring 2004
Lecture 27	April 22, 2004	Carnegie Mellon University

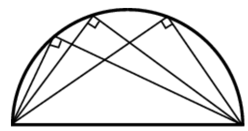
Thales' Legacy: What Is A Proof?

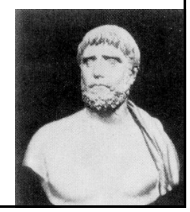


Thales Of Miletus (600 BC) Insisted on Proofs!

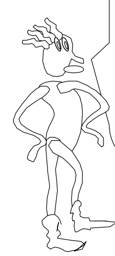
"first mathematician"

Most of the starting theorems of geometry. SSS, SAS, ASA, angle sum equals 180, . . .

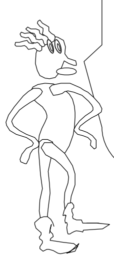




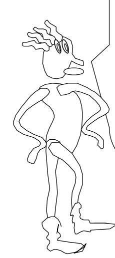
So what is a proof?



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

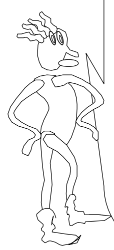


What are "statements"? What does it mean for one to follow "logically" from another?



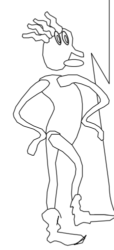
Intuitively, statements must take place in some language.

Formally, statements will take place in a computable language.



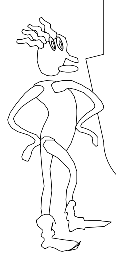
Let S be a computable language over Σ. That is, $S \subset \Sigma^*$ and there is a Java program $P_S(x)$ that outputs Yes if $x \in S$, and outputs No otherwise.

S implicitly defines the "syntactically valid" statements of a language.

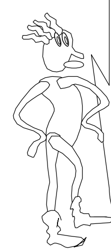


We define our "language" to be a decidable set of strings S. Any $s \in S$ is called a STATEMENT or a SYNTACTICALLY VALID string.

Before pinning down the notion of "logic", let's see some examples of languages in mathematics.



In fact, valid language syntax is typically defined inductively, so it is easy to make a recursive program to recognize the strings considered valid.



Example:
Let S be the set of all syntactically well formed statements in Peano Arithmetic.

$\forall x\ Sx = x$

$\forall x\ Sx = SX$

But not: $===\forall$

Valid Peano Syntax.

Exp -> 0 | S (Exp)

V = $x_1, x_2, x_3, \ldots$

Statement ->

E=E

$\exists$ V (Statement)

$\forall$V (statement)

(statement) Æ (statement)

$\neg$(statement)

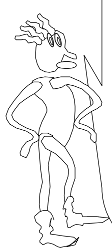
Recursive Program

ValidPeano(s)

return True if any of the following:

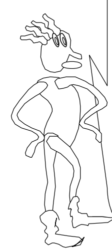
S has form $\forall$ x (T) and Valid$_p$roof(T)

S has form .....



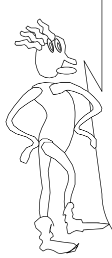
Example:
Let S be the set of all syntactically well formed statements in propositional logic.

X Ç ¬X
XY Æ Y
But not: Æ X Æ Y

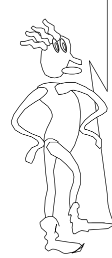


Example:
Let S be the set of all syntactically well formed statements in first-order logic.

$\forall x\ P(x)$
$\forall x \exists y \forall z\ f(x,y,z) = g(x,y,z)$

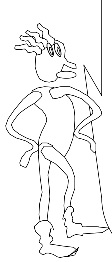


Example:
Let S be the set of all syntactically well formed statements Euclidean Geometry.

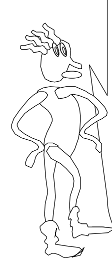


Now we have a way to precisely define what it means to set forth a syntactically valid set of statements in a "language".

What is "logic" and what is "meaning"?



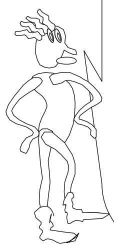
In fact, we will continue to ignore "meaning" and pin down our concepts in purely symbolic (syntactic) terms.



We have a computable set of "statements" S.

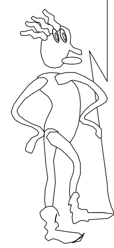
Fix any single computable logic function: $\text{Logic}_S(x,y)$ = Yes/No

If Logic(x,y) = Yes, we say that y is implied by x.



In fact, let's expand the inputs space of our logic function to include a "start statement" Δ not in S.

$Logic_S(\Delta,S)$ = Yes
will mean that our logic views S as an axiom.

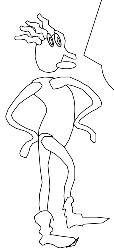


A sequence of statements $s_1, s_2, \ldots, s_n$ is a VALID PROOF of statement Q in $LOGIC_S$ iff

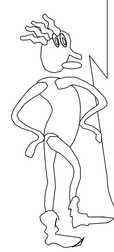
$LOGIC(\Delta, s_1)$ = True

And for n+1> i>1
$LOGIC(s_{i-1},s_i)$ = True

s_n = Q



Notice that our notion of "valid proof" is purely symbolic. In fact, we can make a proofcheck machine to read it and gives a VLID/INVALID answer.



Let S be a set of statements. Let L be a logic function.

$PROVABLE_{S,L}$ =

All $Q \in S$ for which there is a valid proof of Q in logic L

Example: SILLY FOO FOO 1

S = All strings.
L = All pairs of the form: $\langle\Delta, s\rangle$ $s \in S$

$PROVABLE_{S,L}$ is the set of all strings.

Example: SILLY FOO FOO 2

S = All strings.
L = $\langle\Delta, 0\rangle$, $\langle\Delta, 1\rangle$, and
All pairs of the form: $\langle s,s0\rangle$ or $\langle s, s1\rangle$

$PROVABLE_{S,L}$ is the set of all strings.

Example: SILLY FOO FOO 3

S = All strings.

L = $\langle \Delta, 0 \rangle$, $\langle \Delta, 11 \rangle$, and

All pairs of the form:

$\langle s, s0 \rangle$ or $\langle st, s1t1 \rangle$

$\text{PROVABLE}_{S,L}$ is the set of all strings with a zero parity.

Example: SILLY FOO FOO 4

S = All strings.

L = $\langle \Delta, 0 \rangle$, $\langle \Delta, 1 \rangle$, and

All pairs of the form:

$\langle s, s0 \rangle$ or $\langle st, s1t1 \rangle$

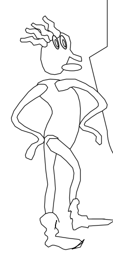
$\text{PROVABLE}_{S,L}$ is the set of all strings.

Example: Propositional Logic

S = All well-formed formulas in the notation of Boolean algebra.

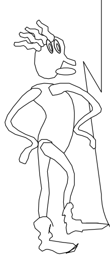
L = Two formulas are one step apart if one can be made from the other from a finite list of forms.

(hopefully) $\text{PROVABLE}_{S,L}$ is the set of all formulas that are tautologies in propositional logic.



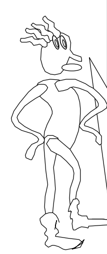
We know what valid syntax is, what logic, proof, and theorems are ....

Where does “truth” and “meaning” come in it?

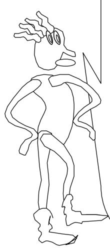


Let S be any computable language. Let TRUTH_S be any fixed function from S to {T, F}.

We will say that we have a “truth concept” associated with the strings in S.

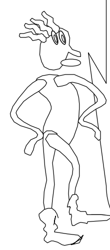


The world of mathematics has certain established truth concepts associated with logical statements.



Let $A(x_1, x_2, .., x_n)$ be a syntactically valid Boolean proposition.

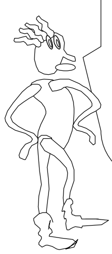
$\text{TRUTH}_{\text{prop logic}}$ (A) is T iff any setting of the variables evaluates to true. A would be called a tautology.



GENERAL PICTURE:

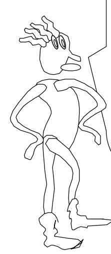
A decidable set of statements S.

A (possibly incomputable) Truth concept TRUTH_S: S → {True, False}

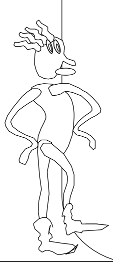


We work in logics that we think are related to our truth concepts.....

A (possibly incomputable) Truth concept TRUTH_S: S → {True, False}



A logic is "sound" for a truth concept if everything it proves is true according to the truth concept.



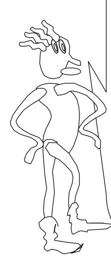
L is SOUND for TRUTH_S if

$L(\Delta, A)$ = true

$\Rightarrow$TRUTH(A)= True

L(B,C)=true and TRUTH(B)=True

$\Rightarrow$TRUTH(C)=True

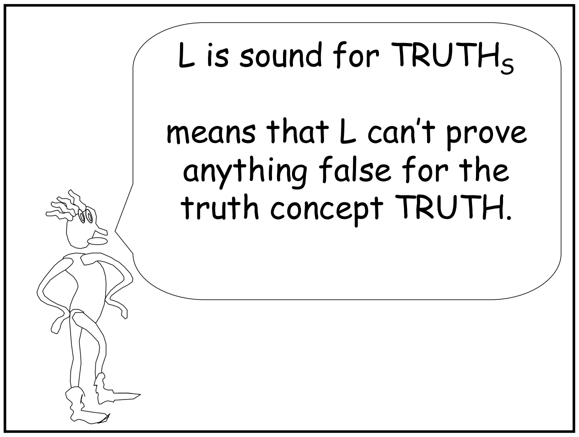


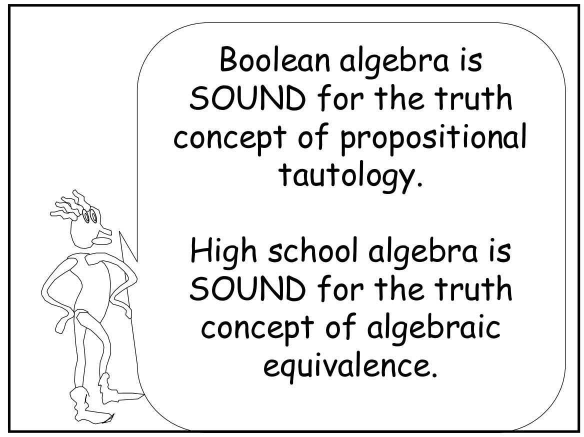
If L is sound for TRUTH_S

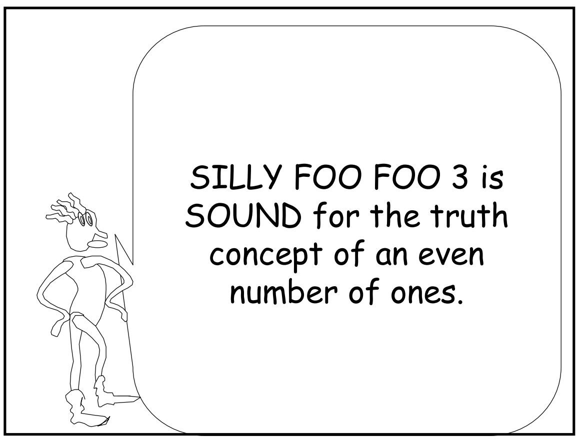
Then

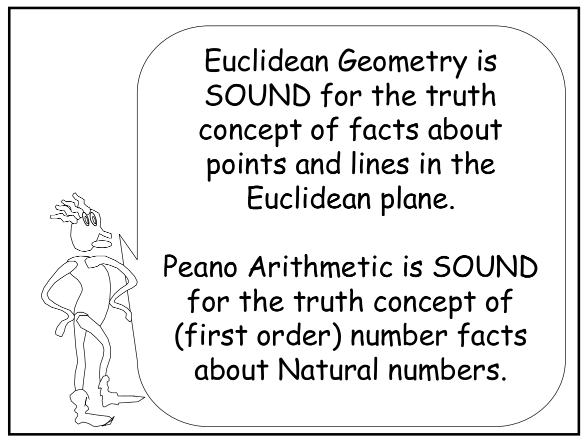
L proves C

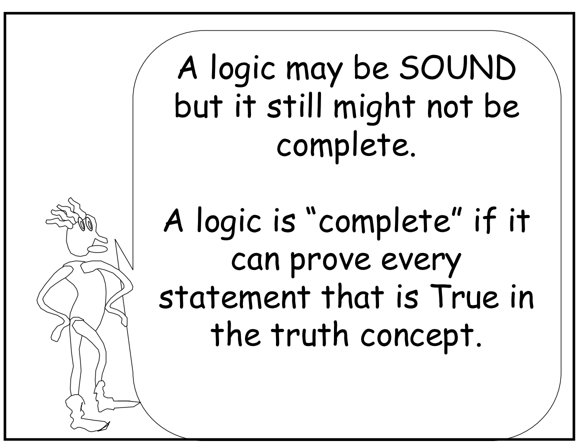
$\Rightarrow \text{TRUTH}_S(C)$ = yes

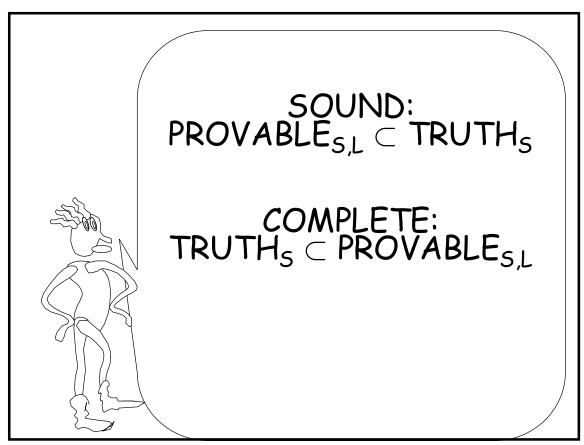

L is sound for $TRUTH_S$
means that L can't prove anything false for the truth concept TRUTH.

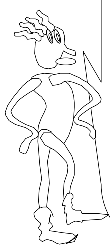

Boolean algebra is SOUND for the truth concept of propositional tautology.
High school algebra is SOUND for the truth concept of algebraic equivalence.


SILLY FOO FOO 3 is SOUND for the truth concept of an even number of ones.


Euclidean Geometry is SOUND for the truth concept of facts about points and lines in the Euclidean plane.
Peano Arithmetic is SOUND for the truth concept of (first order) number facts about Natural numbers.


A logic may be SOUND but it still might not be complete.
A logic is "complete" if it can prove every statement that is True in the truth concept.

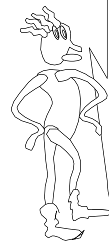

SOUND:
$PROVABLE_{S,L} \subset TRUTH_S$
COMPLETE:
$TRUTH_S \subset PROVABLE_{S,L}$



SOUND:
$\text{PROVABLE}_{S,L} \subset \text{TRUTH}_S$

COMPLETE:
$\text{TRUTH}_S \subset \text{PROVABLE}_{S,L}$

Ex: Axioms of Euclidean Geometry are known to be sound and complete for the truths of line and point in the plane.



SOUND:
$\text{PROVABLE}_{S,L} \subset \text{TRUTH}_S$

COMPLETE:
$\text{TRUTH}_S \subset \text{PROVABLE}_{S,L}$

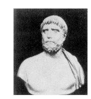
SILLY FOO FOO 3 is sound and complete for the truth concept of strings having an even number of 1s.

Example: SILLY FOO FOO 3

S = All strings.
L = <Δ, 0> , <Δ, 11>, and
All pairs of the form:
<s,s0> or <st, s1t1>

$\text{PROVABLE}_{S,L}$ is the set of all strings with a zero parity.

What is a proof?

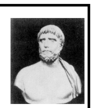


A language.

A truth concept.

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

What is a proof?



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

An ENUMERABLE list of PROVABLE THEOREMS in the logic.

A set S is <u>Recursively Enumerable</u> if its elements can be printed out by a computer program.

In other words:

There is a program LIST_S that outputs a list of strings separated by spaces, and such that an element is on the list if and only if it is in S.

SUPER IMPORTANT

Let F be any logic.

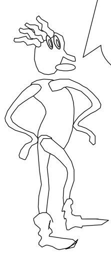
We can write a program to enumerate the provable theorems of F.

Listing THEOREMS$_F$

```
k;=0;

For sum = 0 to forever do

{Let PROOF loop through all strings of length k do
      {Let STATEMENT loop through strings of length <k do
       If proofcheck(STATEMENT, PROOF) = valid, output STATEMENT
      k++
       }
}
```



Whatever the details of our proof system, an inherent property of any proof system is that its theorems are recursively enumerable

Recall: SELF-REFERENCE

Theorem: God is not omnipotent.

Proof: Let S be the statement "God can't make a rock so heavy that he can't lift it.". If S is true, then there is something God can't do, and is hence not omnipotent. If S is false, then God can't lift the rock

Alan Turing (1912-1954)

THEOREM: There is no program to solve the halting problem (Alan Turing 1937)

Suppose a program HALT, solving the halting problem, existed:

HALT(P)= yes, if P(P) halts
HALT(P)= no, if P(P) does not halt

We will call HALT as a subroutine in a new program called CONFUSE.

CONFUSE(P):
If HALT(P) then loop_for_ever
Else return (i.e., halt)
<text of subroutine HALT goes here>

Does CONFUSE(CONFUSE) halt?

YES implies HALT(CONFUSE) = yes
thus, CONFUSE(CONFUSE) will not halt

NO implies HALT(CONFUSE) = no
thus, CONFUSE(CONFUSE) halts

CONFUSE(P):
If HALT(P) then loop_for_ever
Else return (i.e., halt)
<text of subroutine HALT goes here>

Does CONFUSE(CONFUSE) halt?

YES implies HALT(CONFUSE) = yes
thus, CONFUSE(CONFUSE) will not halt

CONTRADICTION

NO implies HALT(CONFUSE) = no
thus, CONFUSE(CONFUSE) halts

K = { P | P(P) halts }

K is an undecidable set. There is no procedure running on an ideal machine to give yes/no answers for all questions of the form "$x \in K$?"

Self-Reference Puzzle

Write a program that prints itself out as output. No calls to the operating system, or to memory external to the program.

Auto_Cannibal_Maker

Write a program AutoCannibalMaker that takes the text of a program EAT as input and outputs a program called $SELF_{EAT}$. When $SELF_{EAT}$ is executed it should output $EAT(SELF_{EAT})$.

For any (input taking) program: EAT
AutoCannibalMaker(EAT) = $SELF_{EAT}$

$SELF_{EAT}$ is a program taking no input.
When executed $SELF_{EAT}$ should output $EAT(SELF_{eat})$

Auto Cannibal Maker
Suppose Halt with no input was programmable in JAVA.

Write a program AutoCannibalMaker that takes the text of a program EAT as input and outputs a program called $SELF_{EAT}$. When $SELF_{EAT}$ is executed it should output $EAT(SELF_{EAT})$

Let EAT(P) = halt, if P does not halt
loop forever, otherwise.

What does $SELF_{EAT}$ do?

Contradiction! Hence EAT does not have a corresponding JAVA program.

Theorems of F

Define the set of provable theorems of F to be the set:

THEOREMS_F =
{ STATEMENT $\in \Sigma^*$ |
$\exists$ PROOF $\in \Sigma^*$,
 proofcheck_F(STATEMENT, PROOF) = valid }

Example: Euclid and ELEMENTS.

We could write a program ELEMENTS to check STATEMENT, PROOF pairs to determine if PROOF is a sequence, where each step is either one logical inference, or one application of the axioms of Euclidian geometry.

$\text{THEOREMS}_{\text{ELEMENTS}}$ is the set of all statement provable from the axioms of Euclidean geometry.

Example: Set Theory and SFC.

We could write a program ZFC to check STATEMENT, PROOF pairs to determine if PROOF is a sequence, where each step is either one logical inference, or one application of the axioms of Zermilo Frankel Set Theory, as well as, the axiom of choice.

$\text{THEOREMS}_{\text{ZFC}}$ is the set of all statement provable from the axioms of set theory.

Example: Peano and PA.

We could write a program PA to check STATEMENT, PROOF pairs to determine if PROOF is a sequence, where each step is either one logical inference, or one application of the axioms of Peano Arithmetic

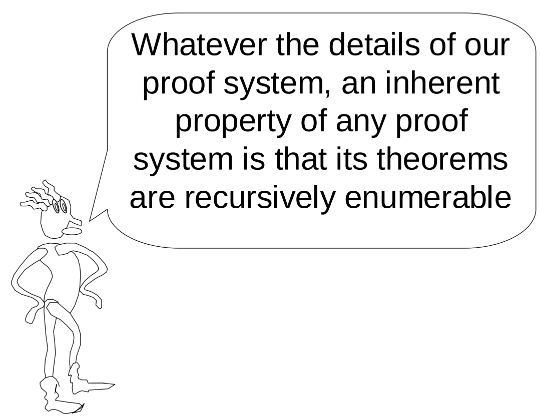
THEOREMS_{PA} is the set of all statement provable from the axioms of Peano Arithmetic

Listing THEOREMS_F

```
k;=0;

For sum = 0 to forever do

{Let PROOF loop through all strings of length k do
      {Let STATEMENT loop through strings of
length <k do
       If proofcheck(STATEMENT, PROOF) = valid,
output STATEMENT
      k++
       }
}
```



Language and Meaning

By a language, we mean any syntactically defined subset of Σ^*

By truth value, we mean a SEMANTIC function that takes expressions in the language to TRUE or FALSE.

Truths of Natural Arithmetic

ARITHMETIC _TRUTH =

All TRUE expressions of the language of arithmetic (logical symbols and quantification over Naturals).

Truths of Euclidean Geometry

EUCLID _TRUTH =

All TRUE expressions of the language of Euclidean geometry.

Truths of JAVA program behavior.

JAVA _TRUTH =

All TRUE expressions of the form program P on input X will output Y, or program P will/won't halt.

TRUTH versus PROVABILITY

Let L be a language L, with a well defined truth function.

If proof system F proves only true statements in the language, we say that F is SOUND.

If F proves all statements in language, we say that F is COMPLETE.

TRUTH versus PROVABILITY

Happy News:

$\text{THEOREMS}_{\text{ELEMENTS}}$ = EUCLID_TRUTH

The ELEMENTS are SOUND and COMPLETE for geometry.

TRUTH versus PROVABILITY

THEOREMS$_{PA}$ is a proper subset of
ARITHMETIC_TRUTH

PA is SOUND.
PA is not COMPLETE.

TRUTH versus PROVABILITY

FOUNDATIONAL CRISIS: It is impossible to have a proof system F such that

THEOREMS$_F$ =
ARITHMETIC_TRUTH

F is SOUND will imply F is INCOMPLETE for arithmetic.

JAVA_TRUTH is not R.E.

Assume a program LIST enumerates JAVA_TRUTH.

We can now make a program Halt(P)

Run list until one of the two statements appears: "P(P) halts", or "P(P) does not halt". Output the appropriate answer.

Contradiction of undecidability of K.

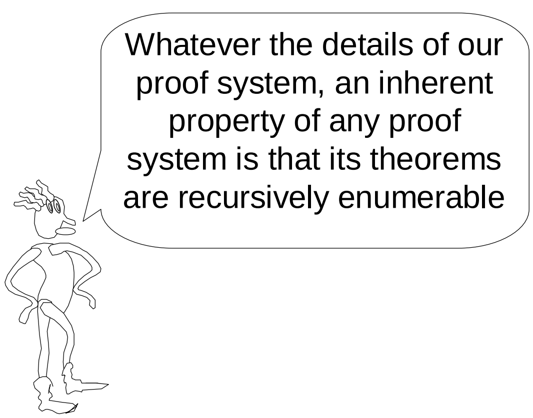
JAVA_TRUTH has no proof system..

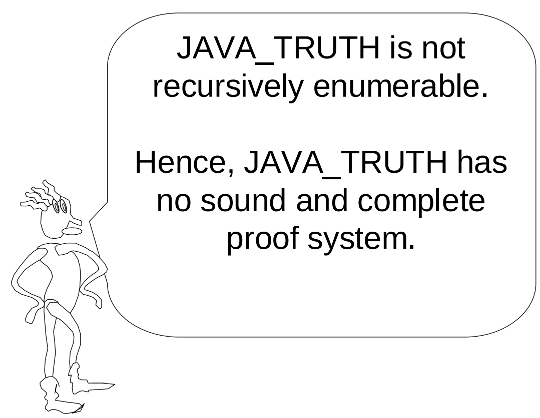
There is no proof system for JAVA$_T$RUTH.

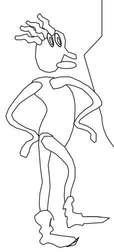
Let F be any proosystem. There must be a program LIST to enumerate THROEMS$_F$.

THEOREM$_F$ is R,E.
JAVA_TRUTH is not R.E.

So THEOREMS$_F$ ≠ JAVA_TRUTH







ARITHEMTIC_TRUTH is not recursively enumerable.

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

Hilbert's 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.

Foundation F

Let F be any foundation for mathematics:

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

INCOMPLETENESS

Let F be any attempt to give a foundation for mathematics

We will 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 symbols S

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

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

$GODEL_F$

$GODEL_F$= AUTO_CANNIBAL_MAKER($CONFUSE_F$)

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

$CONFUSE_F(GODEL_F)$

$GODEL_F$

Can F prove $GODEL_F$ halts?

Yes -> $CONFUSE_F(GODEL_F)$ does not halt
Contradiction

Can F prove $GODEL_F$ does not halt?

Yes -> $CONFUSE_F(GODEL_F)$ halts
Contradiction

$GODEL_F$

F can't prove or disprove that $GODEL_F$ halts.

$GODEL_F$ = $CONFUSE_F(GODEL_F)$
Loop though all sequences of symbols S

If S is a valid F-proof of "$GODEL_F$ halts", then LOOP_FOR_EVER

If S is a valid F-proof of "$GODEL_F$ never halts", then HALT

$GODEL_F$

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.

CONCLUSION

No fixed set of assumptions F can provide a complete foundation for mathematical proof. In particular, it can't prove the true statement that $GODEL_F$ does not halt.

Gödel/Turing: Any statement S of the form "Program P halts on input x" can be easily translated to an equivalent statement S' in the language of Peano Arithmetic. I.e, S is true if and only if S' is true.

Hence: No mathematical domain that contains (or implicitly expresses) Peano Arithmetic can have a complete foundation.

GÖDEL'S INCOMPLETENESS THEOREM

In 1931, 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.

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!

ENDNOTE

You might think that Gödel's theorem proves that are mathematically capable in ways that computers are not. This would show that the Church-Turing Thesis is wrong.

Gödel's theorem proves no such thing!