

15-251

Great Theoretical Ideas in Computer Science



Thales' and Gödel's Legacy: Proofs and Their Limitations

Lecture 27 (April 14, 2009)



A Quick Recap of the Previous Lecture

The Halting Problem $K = \{P \mid P(P) \text{ halts} \}$

Is there a program HALT such that:

- $\text{HALT}(P) = \text{yes}$, if $P \in K$
- $\text{HALT}(P) = \text{no}$, if $P \notin K$

(i.e., 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 of strings $S \subseteq \Sigma^*$ decidable or recursive if there is a program P such that

$P(x) = \text{yes}$, if $x \in S$
 $P(x) = \text{no}$, if $x \notin S$

Hence, the halting set K is undecidable

No program can decide membership in K

Computability Theory: New Vocabulary

We call a set of strings $S \subseteq \Sigma^*$ enumerable
or recursively enumerable (r.e.)
if there is a program P such that:

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



Is
the halting set K
enumerable?

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 $K' = \{ \text{Java } P \mid$
 $P(P) \text{ does not halt} \}$

Is K' enumerable?

No! If both K and K' are enumerable,
then K is decidable. (why?)



And on to newer topics*

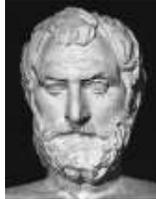
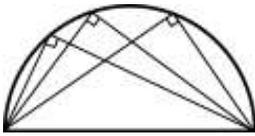
* (The more things change, the more they remain the same...)

What's 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, . . .



Axioms

In traditional logic, an axiom is a proposition that is not proved, but is considered to be self-evident.

Its truth is taken for granted.

It serves as a starting point for inferring other truths.

Peano's Axioms for Arithmetic

The Peano axioms formally define the properties of the natural numbers

1. For every natural number n , $n = n$
2. For all natural numbers, if $n = m$, then $m = n$.
3. For all naturals if $k = m$ and $m = n$ then $k = n$.
4. If n is a natural number and $n = m$, then m is also a natural number.

Peano Arithmetic (contd.)

5. 0 is a natural number.
6. For every natural number n , its “successor” $S(n)$ is a natural number.
7. For every natural number n , $S(n) \neq 0$.
8. For all natural numbers m and n , if $S(m) = S(n)$, then $m = n$.

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?

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

Intuitively, statements are strings in some language.

Formally, statements are strings of a decidable language S over Σ .

That is, S is a subset of Σ^* and there is a Java program $P_S(x)$ that outputs Yes if x is in S , and outputs No otherwise.

This decidable set S is the set of “syntactically valid” strings, or “statements” of a language.

Example:

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

$X \vee \neg X$
 $(X \wedge Y) \Rightarrow Y$
 $\vee X \neg Y$ (not well-formed)

E.g.: syntax for Statements in Propositional Logic

Variable $\rightarrow X, Y, X_1, X_2, X_3, \dots$

Literal \rightarrow Variable | \neg Variable

Statement \rightarrow

Literal

\neg (Statement)

Statement \wedge Statement

Statement \vee Statement

E.g. recursive program to decide S

```
ValidProp(S) {
  return True if any of the following:
```

S has the form $\neg(S_1)$ and $\text{ValidProp}(S_1)$

S has the form $(S_1 \wedge S_2)$ and

$\text{ValidProp}(S_1)$ AND $\text{ValidProp}(S_2)$

S has the form

```
}
```

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

“Statements”: some set S of strings whose membership is decidable.

But what does it mean to follow “logically”? What is “logic”?

Define a function Logic_S

Given a decidable set of statements S , fix a computable “logic function”:

$\text{Logic}_S: (\underline{S} \cup \underline{\Delta}) \times \underline{S} \rightarrow \text{Yes/No}$

If $\text{Logic}(x, y) = \text{Yes}$, we say that the statement y is implied by statement x .

We also have a “start statement” Δ not in S , where $\text{Logic}_S(\Delta, x) = \text{Yes}$ will mean that our logic views the statement x as an axiom.

A valid proof in logic Logic_S

A sequence s_1, s_2, \dots, s_n of statements is a valid proof of statement Q in Logic_S iff

- $\text{Logic}_S(\Delta, s_1) = \text{True}$
(s_1 must be an axiom of our language)
- For all $1 \leq i \leq n-1$, $\text{Logic}_S(s_i, s_{i+1}) = \text{True}$
(each statement implied by the previous one)
- And finally, $s_n = Q$
(the final statement is what we wanted to prove.)

Provable Statements (a.k.a. Theorems)

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

Define $\text{Provable}_{S,L} =$
All statements Q in S for which
there is a valid proof of Q in logic L .

Example: SILLY₁

$S =$ All strings over $\{0,1\}$.
 $L =$ All pairs of the form: $\langle \Delta, s \rangle, s \in S$

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

Example: SILLY₂

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

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

Example: SILLY₃

$S =$ All strings over $\{0,1\}$.
 $L = \langle \Delta, 0 \rangle, \langle \Delta, 11 \rangle,$ and
all pairs of the form: $\langle s, s0 \rangle$ or $\langle s, s11 \rangle$

$\text{Provable}_{S,L}$ is the set of all strings where
all the 1s occur in pairs.

Example: SILLY₄

$S =$ All strings over $\{0,1\}$.
 $L = \langle \Delta, 0 \rangle, \langle \Delta, 11 \rangle,$ 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 an
even number of 1s

More Practical Example: Propositional Logic

$S =$ All well-formed formulas in the notation of
Propositional Logic.

$L =$ Two formulas are one step apart if one
can be made from the other from a
finite list of forms.
(see next page for a partial list.)

Modus ponens
 $[p \rightarrow q] \wedge p \rightarrow [q]$
Modus tollens
 $[p \rightarrow q] \wedge \neg q \rightarrow \neg p$
Constructive introduction (or conjunction)
 $[p] \wedge [q] \rightarrow [p \wedge q]$
Disjunctive introduction (or addition)
 $[p] \rightarrow [p \vee q]$
Simplification
 $[p \wedge q] \rightarrow [p]$
Disjunctive syllogism
 $[p \vee q] \wedge \neg p \rightarrow [q]$
Hypothetical syllogism
 $[p \rightarrow q] \wedge [q \rightarrow r] \rightarrow [p \rightarrow r]$
Constructive dilemma
 $[p \rightarrow q] \wedge [r \rightarrow s] \wedge [p \vee r] \rightarrow [q \vee s]$
Destructive dilemma
 $[p \rightarrow q] \wedge [r \rightarrow s] \wedge [\neg q \vee \neg s] \rightarrow [\neg p \vee \neg r]$
(The same as 2 applications of transposition, then 1 application of constructive dilemma.)
Resolution
 $[p \vee q] \wedge [\neg p \vee r] \rightarrow [q \vee r]$
 ...

More Practical Example: Propositional Logic

S = All well-formed formulas in the notation of Propositional Logic.

L = Two formulas are one step apart if one can be made from the other from a finite list of forms.
(see previous page for a partial list.)

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

Super-important fact about theorems

Let **S** be a decidable set of statements.
Let **L** be a computable logic function.

Define $\text{Provable}_{S,L} =$
All statements **Q** in **S** for which
there is a valid proof of **Q** in logic **L**.

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

I.e., $\text{Provable}_{S,L}$ is enumerable.

Enumerating the Set $\text{Provable}_{S,L}$

```

for k = 0 to forever do
{
  let PROOF loop through all strings of length k
  {
    let STMT loop through all strings of length < k
    {
      if proofcheckS,L(STMT, PROOF) = Valid
      {
        output STMT;           //this is a theorem
      }
    }
  }
}

```

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 (enumerable) set of all statements provable from the axioms of Euclidean geometry.

Example: Peano and Peano-Arith.

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

$\text{THEOREMS}_{\text{PeanoA}}$ is the (enumerable) set of all statements provable from the axioms of Peano Arithmetic.

OK, so I see
what valid syntax is, ← *decidable*
what a logic is, ← *computable*
what a proof is,
what theorems are (and that
they can be enumerated)...

But where does “truth” and
“meaning” come in it?

Let S be any decidable set of
statements

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 .

This is a very general definition, we
can define all kinds of truths:

E.g.: $\text{AllTruth}_S(x) = \text{True}$ for all $x \in S$

E.g.: $\text{NoTruth}_S(x) = \text{False}$ for all $x \in S$

E.g.: $\text{OddTruth}_S(x) = \text{True}$
for all $x \in S$ that have even length

These are pretty useless truth concepts, we
will deal with more useful truth concepts...

Truths of Euclidean Geometry

EuclidTruth =

All TRUE expressions of the
language of Euclidean
geometry.

Truths of Natural Arithmetic

ArithmeticTruth =

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

Truths of JAVA Program Behavior

JavaTruth =

All TRUE expressions of the form
program “P on input X will halt” or
“P on input X will not halt”

General Picture

A decidable set of statements S .

A computable logic L .

A (possibly uncomputable)
truth concept
 $\text{Truth}_S: S \rightarrow \{T, F\}$

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

A logic L is “sound” for a truth concept Truth_S if

$$x \text{ in Provable}_{S,L} \Rightarrow \text{Truth}_S(x) = T$$

L is sound for Truth_S if

- $L(\Delta, A) = \text{true} \Rightarrow \text{Truth}_S(A) = \text{True}$

- $L(B,C) = \text{True} \text{ and } \text{Truth}_S(B) = \text{True} \Rightarrow \text{Truth}_S(C) = \text{True}$

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

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

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

In other words, $\text{Provable}_{L,S} \Rightarrow \text{Truth}_S$

SILLY_3 is sound for the truth concept of an even number of ones.

Example SILLY_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 zero parity.

Euclidean Geometry is sound for the EuclidTruth (the truth concept of facts about points and lines in the Euclidean plane)

Peano Arithmetic is sound for the ArithmeticTruth , the truth concept of (first order) number facts about naturals.

A logic may be sound but it still might not be “complete”

A logic L is complete for a truth concept Truth_S if it can prove every statement that is True in Truth_S

Soundness:
 $\text{Provable}_{S,L} \Rightarrow \text{Truth}_S$

Completeness:
 $\text{Truth}_S \Rightarrow \text{Provable}_{S,L}$

SILLY_3 is sound and complete for the truth concept of an even number of ones.

Example SILLY_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 an even number of 1's.

Truth versus Provability

Happy News:

$\text{Provable}_{\text{Elements}} = \text{EuclidTruth}$

The Elements of Euclid are sound and complete for (Euclidean) geometry.

Truth versus Provability

Happy News:

$\text{Provable}_{\text{Boolean}} = \text{PropositionalTruth}$

The rules of Boolean Algebra are sound and complete for propositional tautologies.

Truth versus Provability

Harsher Fact:

$\text{Provable}_{\text{PeanoArith}}$ is a proper subset of ArithmeticTruth

Peano Arithmetic is sound.

It is not complete.



Truth versus Provability

Foundational Crisis:

It is impossible to have logic L such that
 $\text{Provable}_{S,L} = \text{ArithmeticTruth}$

If F is sound for arithmetic, then F cannot be complete.



Let's prove this...

Ingredients:

A language S.

A truth concept Truth_S .

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

(Hence) an enumerable list $\text{Provable}_{S,L}$ of theorems in the logic.

JavaTruth is Not Enumerable

Suppose JavaTruth is enumerable, and the program JavaLIST enumerates JavaTruth.

Can now make a program HALT(P):

Run JavaLIST until either of the two statements appears: "P(P) halts", or "P(P) does not halt".

Output the appropriate answer.

So HALT(P) would decide halting for P(P) in finite time.

This contradicts the undecidability of halting set K.

Hence: JavaTruth has no sound and complete proof system

Suppose there is. Then there must be a program to enumerate $\text{Provable}_{S,L} = \text{Truth}_S$

$\text{Provable}_{S,L}$ is recursively enumerable.

By previous slide, JavaTruth is not recursively enumerable.

So $\text{Provable}_{S,L} \neq \text{JavaTruth}$

To Recap:

The Halting problem is not decidable.

Hence, JavaTruth is not recursively enumerable.

Hence, JavaTruth has no sound and complete proof system.

Corollary: ArithmeticTruth has no sound and complete proof system

Given a program P, we can build a Diophantine equation which has an integer root iff P(P) halts. } non-trivial

Hence the existence of integer roots for Diophantine equations is not decidable.

Hence, ArithmeticTruth is not recursively enumerable.

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

Foundational Crisis

If F is sound for arithmetic, then F cannot be complete.



Some Historical Context

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

Gödel's Incompleteness Theorem

In 1931, Kurt Gödel stunned the world by proving that for any foundation of mathematics 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.

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.)

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.

GODEL_F = AutoCannibalMaker(CONFUSE_F)

Running GODEL_F 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 =
AutoCannibalMaker(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

Can F prove GODEL_F does not halt?

If Yes, then CONFUSE_F(GODEL_F) halts:
Contradiction

GODEL_F

F can't prove or disprove that GODEL_F halts.

But GODEL_F = CONFUSE_F(GODEL_F) is the
program:

Loop though all sequences of sentences in S

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

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

Since F can't prove/disprove GODEL_F halts,
this program GODEL_F will loop forever!

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.



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!

