15-251

Great Theoretical Ideas in Computer Science

Gödel's Legacy: Proofs and Their Limitations

Lecture 25 (November 16, 2010)

A Quick Recap of the Previous Lecture

The Halting Problem

Is there a program HaltsOnItself such that
for all programs P:

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

Halting Set K = { Java P | P(P) halts }

Alan Turing (1912-1954)

Theorem: [1937]

There is no program to solve the halting problem

CONFUSE

Suppose HaltsOnItself exists

```
CONFUSE(P)
{ if (HaltsOnItself(P))
      then loop forever;      //i.e., we dont halt
  else exit;                  //i.e., we halt
  // text of HaltsOnItself goes here
}
```

Does CONFUSE(CONFUSE) halt?

Alan Turing (1912-1954)

Theorem: [1937]

There is no program to solve the halting problem

Note: The impossibility proof even holds for "good" programs P with

1. at most one (clearly specified) input stmt
2. at most one (clearly specified) exit statement

```
PROGRAM
{     read input INP;      // input stmt
      blah blah;
      blah;
      exit;                // exit stmt
}
```

Computability Theory: Vocabulary Lesson

We call a set $S \subseteq \Sigma^*$ decidable or recursive if there is a program P such that:

P(x) = yes, if $x \in S$

P(x) = no, if $x \notin S$

We saw: the halting set K is undecidable

(No program can decide membership in K)

Computability Theory: Some More 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.

We saw: the halting set K is enumerable

The $Halt_0$ Problem

Is there a program Halts such that for all programs Q which take no input:

Halts(Q) = yes, if Q halts

Halts(Q) = no, if Q does not halt

Set K_0 = { Java Q | Q halts }

Claim: The set K_0 is undecidable.

Claim: The set K_0 is undecidable.

Proof: If K_0 decidable, there exists Halts

Using this, we claim we can decide K.

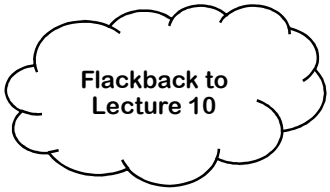
Take any program P

```
P {
      read input INP;      // input stmt
      blah blah;
      exit;                // exit stmt
}
```

Make program Q = P(P)

```
Q {
      var INP = "text of P";
      blah blah;
      exit;                // exit stmt
}
```

Now can use Halts to decide K. Contradiction!



What's a proof?

What is a proof?

A sequence of statements,
each of which

is an axiom,

or a hypothesis,

or follows from previous statements
using an inference rule

A Logical System

A "logic" consists of:

1) A collection S of well-formed sentences
2) Some sentences in S called "axioms"
3) A set of "rules of inference"

E.g., Logical System for Propositions

Axiom:

$$(\neg A \vee A)$$

Inference Rules:

$$\frac{A \vee (B \vee C)}{(A \vee B) \vee C}$$ associativity

$$\frac{A \vee A}{A}$$ contraction

$$\frac{A}{B \vee A}$$ expansion

$$\frac{(A \vee B),\ (\neg A \vee C)}{(B \vee C)}$$ cut rule

A Logical System

A "logic" consists of:

1) A collection S of well-formed sentences
2) Some sentences in S called "axioms"
3) A set of "rules of inference"

A "truth concept" consists of:

1) A collection S of well-formed sentences
2) Some sentences in S called "truths"

Our logical system for propositions is

sound "all theorems are true"

and

complete "all truths are theorems"

for propositional truths (tautologies)

A Logical System

A "logic" consists of:

1) A collection S of well-formed sentences
2) Some sentences in S called "axioms"
3) A set of "rules of inference"

A "truth concept" consists of:

1) A collection S of well-formed sentences
2) Some sentences in S called "truths"

Another example: Peano Arithmetic

a) 0 is a natural number.

b) For every natural number n, its "successor" S(n) is a natural number.

c) For every natural number n, $S(n) \neq 0$.

d) For all natural numbers m and n, if $S(m) = S(n)$, then $m = n$.

Peano Arithmetic (contd.)

e) For every natural n, n = n

f) For all naturals, if n = m, then m = n.

g) For all naturals if k = m and m = n then k = n.

h) If n is a natural number and n = m, then m is also a natural number.

Peano: and so on…

You can build an edifice for arithmetic

and use this logical system

to conceivably prove all arithmetic truths

Thm: the Peano system is sound for arithmetic

We asked: Is it complete?

We'll answer this today…

General Picture

A set of (well-formed) statements S.

A logic L.

A truth concept
Truth_S: $S \rightarrow \{T, F\}$

Extra condition 1

We want the set of statements to be decidable

I.e., there exists an algorithm to check well-formedness

Recursive Program to decide S

(for propositional logic)

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

S has the form $\neg(S_1)$ and ValidProp(S_1)
S has the form $(S_1 \wedge S_2)$ and
ValidProp(S_1) AND ValidProp(S_2)
S has the form .....

}

General Picture

A decidable set of statements S.

A logic L.

A truth concept
$Truth_S$: S $\rightarrow$ {T, F}

Extra condition 2

We want the logic to be computable

I.e., there exists an algorithm to decide:
a) given a statement s, is it an axiom?
b) given a statement s and s', if
s' follows from s using an inference rule

General Picture

A decidable set of statements S.

A computable logic L.

A truth concept
$Truth_S$: S $\rightarrow$ {T, F}

What conditions on truth concept?

None.

This is the elusive "right or wrong" we are trying to capture…

Truths of Propositional Logic

PropositionalTruth =

All expressions in propositional logic that are tautologies.

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 "not halt"

General Picture

A decidable set of statements S.

A computable logic L.

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

Super Important Fact

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

Theorem: We can write a program to enumerate all the theorems of L.

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

Enumerating the Set $Provable_{S,L}$

```
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 proofcheck_{S,L}(STMT, PROOF) = Valid
                output STMT;        //this is a theorem
```

We can enumerate all the theorems of propositional logic, Elements of Euclid, Peano arithmetic! (and all programs in K.)

General Picture

A decidable set of statements S.

A computable logic L.

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

We can enumerate $Theorems_{S,L}$

Soundness:

Every theorem of (S,L) is true (according to $TRUTH_S$)

Completeness:

Every truth (according to $TRUTH_S$) is a theorem of (S,L)

Truth versus Provability

Happy News:

$Provable_{PropLogic}$ = PropositionalTruth

The logical system we gave was sound and complete for Prop.Logic

Truth versus Provability

Happy News:

$Provable_{Elements}$ = EuclidTruth

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

Truth versus Provability

Not-so-Happy News:

$Provable_{Peano} \neq$ ArithmeticTruth

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.

Foundation for mathematics:
F = (statements S, logical system L) + soundness


Hilbert

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.

Truth versus Provability

Foundational Crisis:
It is impossible to have a logical system F such that
$\text{Provable}_{F,S} = \text{ArithmeticTruth}$

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

Here's what we have

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

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

JAVATruth is Not Enumerable

Suppose JAVATruth is enumerable, and the program JavaList enumerates JAVATruth.

Can now make a program HaltsOnItself(P):

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

Contradiction of undecidability of K.

JAVATruth has No Proof System

Theorem: There is no sound and complete proof system for JAVATruth.

Proof:

Suppose (S,L) is sound and complete.
Recall, we can enumerate $\text{Provable}_{S,L}$.
By soundness+completeness,
$\text{Provable}_{S,L} = \text{JAVATruth}$

Contradicts the fact that JAVATruth is not recursively enumerable.

The Halting problem is not decidable.

Hence, JavaTruth is not recursively enumerable.

Hence, JavaTruth does not have a sound+complete logical system.

We can show that the existence of integer roots for Diophantine equations is not decidable.

Polynomials capture the behavior of programs here!!!

Hence, ArithmeticTruth is not recursively enumerable.

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

Incompleteness

Let us fix F=(S,L) 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.

Suppose F is sound+complete for JAVATruth.

Then for each P, F can prove either "P halts" or "P does not halt"

```
CONFUSE(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.

$GODEL_F$ = 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

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.

And this program does not halt!

No fixed set of assumptions F can provide a complete foundation for mathematical proof.

In particular, it can't prove the true statement "$GODEL_F$ 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!