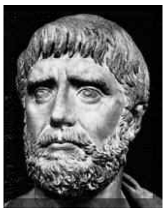
Great Theoretical Ideas In Computer Science		
Anupam Gupta		CS 15-251 Fall 2005
Lecture 27	Dec 1, 2005	Carnegie Mellon University

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



A Quick Recap of the Previous Lecture

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

Is there a program HALT such that:

HALT(P) = yes, if $P \in K$
HALT(P) = no, if $P \notin K$

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

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

Hence, the halting set K is undecidable

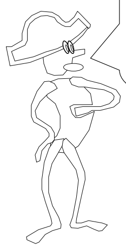
Computability Theory: New Vocabulary

We call a set $S \subseteq \Sigma^*$ 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.

Is
the halting set K
enumerable?



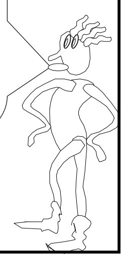
Enumerating K

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

Is K' enumerable?



If both K and K' are enumerable,
then K is decidable. (why?)

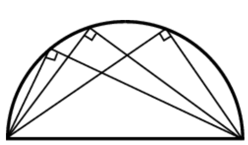
And on to newer topics*

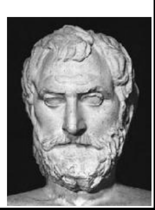
*(The more things change, the more they remain the same…)

Thales Of Miletus (600 BC)
Insisted on Proofs!

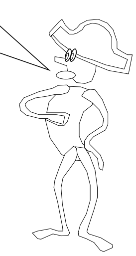
"first mathematician"

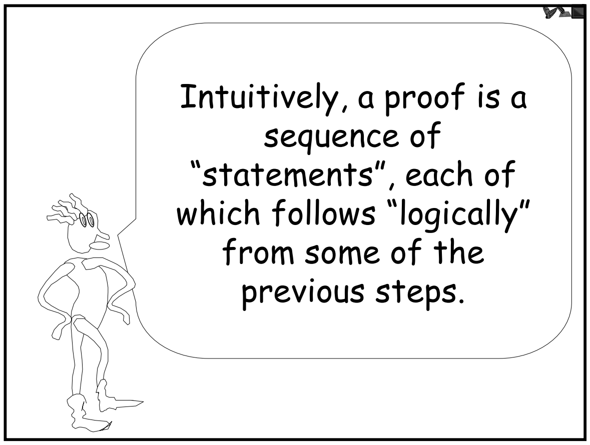
Most of the starting theorems of geometry.
SSS, SAS, ASA, angle sum equals 180, . . .

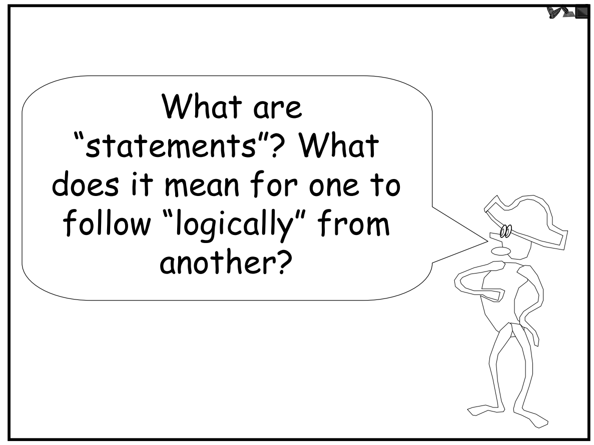


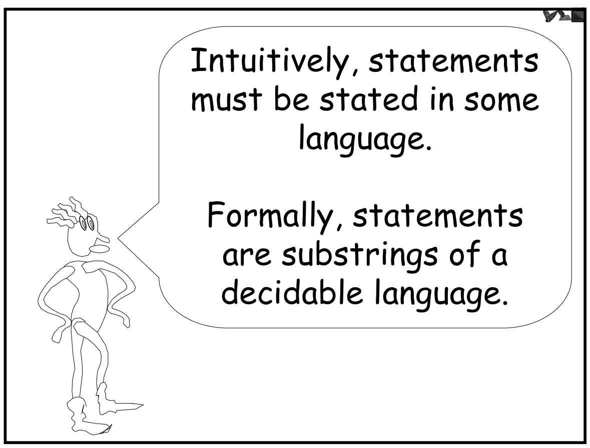


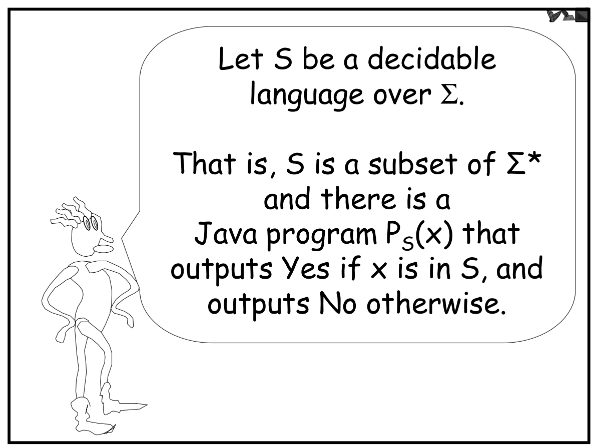
What is a proof
anyways?

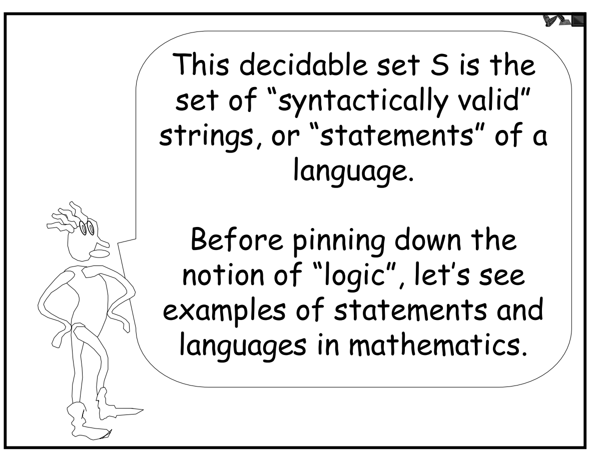


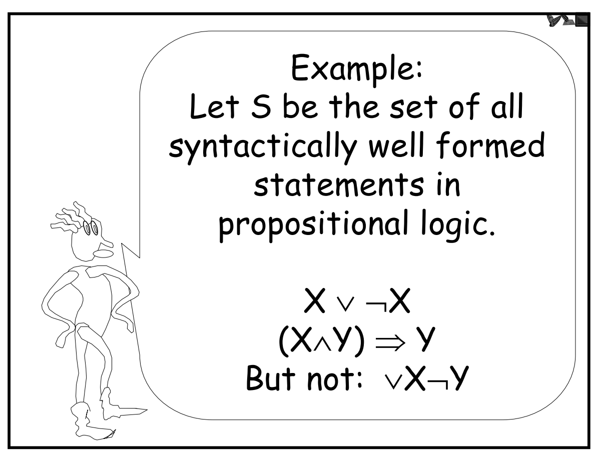

Intuitively, a proof is a sequence of "statements", each of which follows "logically" from some of the previous steps.

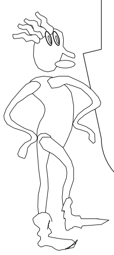

What are "statements"? What does it mean for one to follow "logically" from another?


Intuitively, statements must be stated in some language.
Formally, statements are substrings of a decidable language.


Let S be a decidable language 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.
Before pinning down the notion of "logic", let's see examples of statements and languages in mathematics.


Example:
Let S be the set of all syntactically well formed statements in propositional logic.
$X \vee \neg X$
$(X \wedge Y) \Rightarrow Y$
But not: $\vee X \neg Y$



Typically, valid language syntax is defined inductively.

This makes it easy to write a recursive program to recognize the strings in the language.

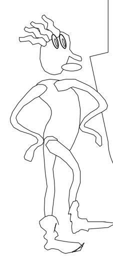
Syntax for Statements in Propositional Logic

Variable $\to$ X, Y, X_1, X_2, X_3, …
Literal $\to$ Variable | $\neg$Variable

Statement $\to$
- Literal
- $\neg$(Statement)
- Statement $\wedge$ Statement
- Statement $\vee$ Statement

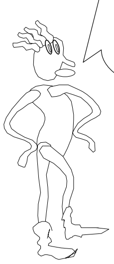
Recursive Program to decide S

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

   S has the form ¬(S1) and ValidProp(S1)
   S has the form (S1 ∧ S2) and
        ValidProp(S1) AND ValidProp(S2)
   S has the form  .....

}
```



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

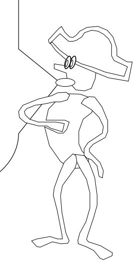
$\forall$ x P(x)
$\forall$ x$\exists$y$\forall$z f(x,y,z) = g(x,y,z)

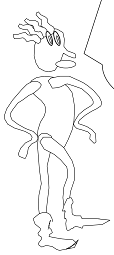


Example:
Let S be the set of all syntactically well formed statements in Euclidean Geometry.

OK, we can now precisely define a syntactically valid set of “statements” in a language.

But what is “logic”, and what is “meaning”?





For the time being, let us ignore the meaning of "meaning", and pin down our concepts in purely symbolic (syntactic) terms.

Define a function $Logic_S$

Given a decidable set of statements S, fix any single computable "logic function":

$Logic_S: (S \cup \Delta) \times S \rightarrow$ Yes/No

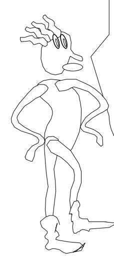
If Logic(x,y) = Yes, we say that the statement y is implied by statement x.

We also have a "start statement" Δ not in S, where $Logic_S(\Delta,x)$ = Yes will mean that our logic views the statement x as an axiom.

A valid proof in logic $Logic_S$

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

- $Logic_S(\Delta, s_1)$ = True
  (i.e., s_1 is an axiom of our language)
- For all $1 \leq j \leq n-1$, $Logic_S(s_j, s_{j+1})$ = True
  (i.e., each statement implies the next one)
- and finally, $s_n = Q$
  (i.e., the final statement is indeed Q.)



Notice that our notion of "valid proof" is purely symbolic.

In fact, we can make a proof-checking machine to read a purported proof and give a Valid/Invalid answer.

Provable Statements (a.k.a. Theorems)

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

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

Example $SILLY_1$

S = All strings.

L = All pairs of the form: $<\Delta, s>$ $s \in S$

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

Example: $SILLY_2$

S = All strings.

L = <Δ, 0> , <Δ, 1>, and
all pairs of the form: <s,s0> or <s, s1>

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

Example: $SILLY_3$

S = All strings.

L = <Δ, 0> , <Δ, 11>, and
all pairs of the form: <s,s0> or <st, s1t1>

$Provable_{S,L}$ is the set of all strings with zero parity.

Example: $SILLY_4$

S = All strings.

L = <Δ, 0> , <Δ, 1>, and
all pairs of the form: <s,s0> or <st, s1t1>

$Provable_{S,L}$ 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. (see next page for a partial list.)

Modus ponens

$[(p \rightarrow q) \wedge p] \rightarrow [q]$

Modus tollens

$[(p \rightarrow q) \wedge \neg q] \rightarrow [\neg p]$

Conjunction introduction (or *Conjunction*)

$[(p) \wedge (q)] \rightarrow [p \wedge q]$

Disjunction 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

$[(p \vee q) \wedge (\neg p \vee r)] \rightarrow [(q \vee r)]$

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) $Provable_{S,L}$ is the set of all formulas that are tautologies in propositional logic.

Super Important Fact

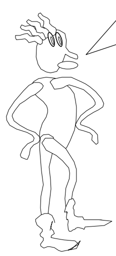
Let S be any (decidable) set of statements.
Let L be any (computable) logic.

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

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

Enumerating the set $Provable_{S,L}$

```
k=0;
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
      }
    }
  }
}
```



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

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.

$THEOREMS_{ELEMENTS}$ 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.

$THEOREMS_{ZFC}$ is the set of all statement provable from the axioms of set theory.

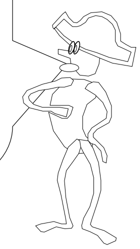
Example: Peano and PA.

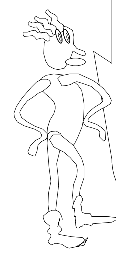
We could write a program PA to check STATEMENT, PROOF pairs to determine if PROOF is a sequence, where each step is either one logical inference, or one application of the axioms of Peano Arithmetic

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

OK, so I see what valid syntax is, what logic is, what a proof and what theorems are…

But where does "truth" and "meaning" come in it?





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.

Truths of Natural Arithmetic

Arithmetic_Truth =

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

Truths of Euclidean Geometry

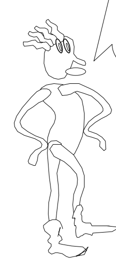
Euclid_Truth =

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.



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

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

$Truth_{prop\ logic}$ (P) is T
iff
any setting of the variables evaluates to true.

P is then called a tautology.

General Picture:

A decidable set of statements S.

A computable logic L.

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

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 in $Provable_{S,L}$
$\Rightarrow Truth_S(x) = T$

L is sound for $Truth_S$ if

$L(\Delta, A)$ = true
$\Rightarrow Truth_S(A)$= True

L(B,C)=true and
$Truth_S(B)$=True
$\Rightarrow Truth_S(C)$= True

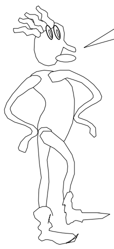
L is sound for $Truth_S$ means that L can't prove anything false for the truth concept $Truth_S$.

i.e.,

$Provable_{L,S} \subseteq Truth_S$

Boolean algebra is sound for the truth concept of propositional tautology.

High school algebra is sound for the truth concept of algebraic equivalence.

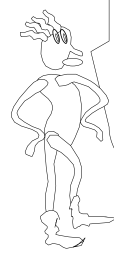


SILLY$_3$ is sound for the truth concept of an even number of ones.

Example SILLY$_3$

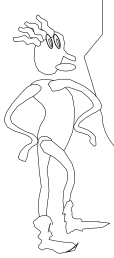
S = All strings.
L = $\langle \Delta, 0\rangle$, $\langle \Delta, 11\rangle$, and
all pairs of the form: $\langle s,s0\rangle$ or $\langle st, s1t1\rangle$

Provable$_{S,L}$ is the set of all strings with zero parity.



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.



However, a logic may be sound but it still might not be “complete”.

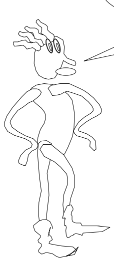
A logic L is complete for a truth concept Truth$_S$ if it can prove every statement that is True in Truth$_S$

Soundness:

$\text{Provable}_{S,L} \subset \text{Truth}_S$

Completeness:

$\text{Truth}_S \subset \text{Provable}_{S,L}$



SILLY$_3$ is sound <u>and</u> complete for the truth concept of strings having an even number of 1s.

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 $\langle st, s1t1\rangle$

Provable$_{S,L}$ is the set of all strings with zero parity.

How about other logics?

Which natural logics are sound <u>and</u> complete?

Truth versus Provability

Happy News:

$Provable_{Elements}$ = Euclid_Truth

The Elements of Euclid are sound <u>and</u> complete for (Euclidean) geometry.

Truth versus Provability

Harsher Fact:

$Provable_{PeanoArith}$ is a <u>proper</u> subset of Arithmetic_Truth

Peano Arithmetic is sound.

It is <u>not complete</u>.

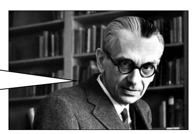
Truth versus Provability

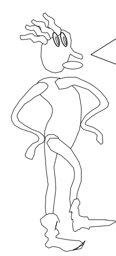
<u>Foundational Crisis</u>:

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.





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

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 $Provable_{S,L}$ of provable statements (theorems) in the logic.

JAVA_Truth is not enumerable

Suppose JAVA_Truth is enumerable, and the program JAVA_LIST enumerates JAVA_Truth.

Can now make a program HALT(P):

Run JAVA_LIST 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.

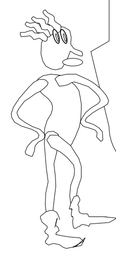
JAVA_Truth has no proof system

There is no sound and complete proof system for JAVA_Truth.

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

$\text{Provable}_{L,S}$ is r.e.
JAVA_Truth is not r.e.

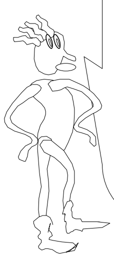
So $\text{Provable}_{L,S} \neq$ JAVA_Truth



The Halting problem is not decidable.

Hence, JAVA_Truth is not recursively enumerable.

Hence, JAVA_Truth has no <u>sound and complete</u> proof system.



Similarly, in the last lecture, we saw that 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

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.

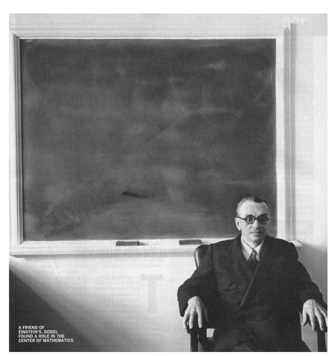
but $GODEL_F = CONFUSE_F(GODEL_F)$ is the program

Loop though all sequences of sentences in S

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

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

but this program does not halt



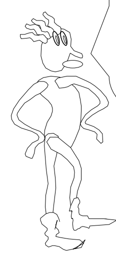
To summarize

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.



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!

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!