CS154, Lecture 11: Self Reference, Foundation of Mathematics
Self-Reference and the Recursion Theorem

1. Living things are machine
2. Living things can self-reproduce
3. Machines cannot self-reproduce

Paradox?
Lemma: There is a computable function $q : \Sigma^* \rightarrow \Sigma^*$ such that for every string $w$, $q(w)$ is the description of a TM $P_w$ that on every input, prints out $w$ and then accepts.

“Proof” Define a TM $Q$:

$Q \xrightarrow{w} P_w (P_w erases its input and then prints $w$)
Theorem: There is a Self-Printing TM

Proof: First define a TM \( B \) which does this:

Now consider the TM that looks like this:
The Recursion Theorem

Theorem: For every TM $T$ computing a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ there is a Turing machine $R$ computing a function $r : \Sigma^* \rightarrow \Sigma^*$, such that for every string $w$,

$$r(w) = t(R, w)$$
For every computable $t$, there is a computable $r$ such that $r(w) = t(R,w)$ where $R$ is a description of $r$

Suppose we can design a TM $T$ of the form:
"On input $(x,w)$, do bla bla with $x$ and $w$, etc. etc."

We can then find a TM $R$ with the behavior:
"On input $w$, do bla bla with a description of $R$ and $w$, etc. etc."

We can use the operation:
"Obtain your own description"
in Turing machine pseudocode!
Theorem: \( A_{\text{TM}} \) is undecidable

Proof (using the recursion theorem)

Assume \( H \) decides \( A_{\text{TM}} \)

Construct machine \( B \) such that on input \( w \):

1. Obtains its own description \( B \)
2. Runs \( H \) on \((B, w)\) and flips the output

Running \( B \) on input \( w \) always does the opposite of what \( H \) says it should!

A formalization of “free will” paradoxes!

No single machine can predict behavior of all others
Computability and the Foundations of Mathematics
A formal system describes a formal language for
- writing (finite) mathematical statements,
- has a definition of what statements are “true”
- has a definition of a proof of a statement

Example: Every TM $M$ defines some formal system $F$
{Mathematical statements in $F$} = $\Sigma^*$ String $w$ represents the statement “$M$ accepts $w$”
✓ {True statements in $F$} = $L(M)$
✓ A proof that “$M$ accepts $w$” can be defined to be an accepting computation history for $M$ on $w$
Interesting Formal Systems

Define a formal system $F$ to be *interesting* if:

1. Mathematical statements that can be precisely described in English should be expressible in $F$

2. Proofs are “convincing” – a TM can check that a proof of a theorem is correct (decidable)

3. Simple proofs that can be precisely described in English should be expressible in $F$
Interesting Formal Systems

Define a formal system $\mathcal{F}$ to be *interesting* if:

1. Any mathematical statement about computation can be (computably) described as a statement of $\mathcal{F}$.
   Given $(M, w)$, there is a (computable) $S_{M,w}$ in $\mathcal{F}$ such that $S_{M,w}$ is true in $\mathcal{F}$ if and only if $M$ accepts $w$.

2. Proofs are “convincing” – a TM can check that a proof of a theorem is correct.
   *This set is decidable: $\{(S, P) \mid P$ is a proof of $S$ in $\mathcal{F}\}$*

3. If $S$ is in $\mathcal{F}$ and there is a proof of $S$ describable as a computation, then there’s a proof of $S$ in $\mathcal{F}$.
   *If $M$ accepts $w$, then there is a proof $P$ in $\mathcal{F}$ of $S_{M,w}$*
A formal system $F$ is **consistent** or **sound** if no false statement has a valid proof in $F$ (Proof in $F$ implies Truth in $F$)

A formal system $F$ is **complete** if every true statement has a valid proof in $F$ (Truth in $F$ implies Proof in $F$)

**Consistency and Completeness**
Limitations on Mathematics

For every consistent and interesting $F$,

Theorem 1. (Gödel 1931) $F$ is incomplete: There are mathematical statements in $F$ that are true but cannot be proved in $F$.

Theorem 2. (Gödel 1931) The consistency of $F$ cannot be proved in $F$.

Theorem 3. (Church-Turing 1936) The problem of checking whether a given statement in $F$ has a proof is undecidable.
Proof: Define Turing machine \( G(x) \):

1. Obtain own description \( G \) [Recursion Theorem]
2. Construct statement \( S' = \neg S_{G,\varepsilon} \)
3. Search for a proof of \( S' \) in \( F \) over all finite length strings. Accept if a proof is found.

Claim: \( S' \) is true in \( F \), but has no proof in \( F \)

\( S' \) basically says “There is no proof of \( S' \) in \( F \)”
(Gödel 1931) The consistency of $F$ cannot be proved within any interesting consistent $F$

Proof: Suppose we can prove “$F$ is consistent” in $F$

We constructed $\neg S_{G, \varepsilon} = “G$ does not accept $\varepsilon”$ which we showed is true, but has no proof in $F$

$G$ does not accept $\varepsilon$ $\iff$ There is no proof of $\neg S_{G, \varepsilon}$ in $F$

But if there’s a proof in $F$ of “$F$ is consistent” then there is a proof in $F$ of $\neg S_{G, \varepsilon}$ (here’s the proof):

“If $S_{G, \varepsilon}$ is true, then there is a proof in $F$ of $\neg S_{G, \varepsilon}$

$F$ is consistent, therefore $\neg S_{G, \varepsilon}$ is true.

But $S_{G, \varepsilon}$ and $\neg S_{G, \varepsilon}$ cannot both be true.

Therefore, $\neg S_{G, \varepsilon}$ is true”

This contradicts the previous theorem.
Proof: Suppose $\text{PROVABLE}_F$ is decidable with TM $P$
Then we can decide $A_{TM}$ using the following procedure:
On input $(M, w)$, run the TM $P$ on input $S_{M,w}$
If $P$ accepts, examine all possible proofs in $F$
If a proof of $S_{M,w}$ is found then accept
If a proof of $\neg S_{M,w}$ is found then reject
If $P$ rejects, then reject.

Why does this work?

(Church-Turing 1936) For every interesting consistent $F$, $\text{PROVABLE}_F$ is undecidable