CS 154

Self-Reference, Computability, and Logic

Next Tuesday (2/14)

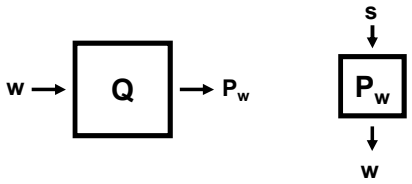
Midterm in class

Sample Midterm coming out today… look for it!

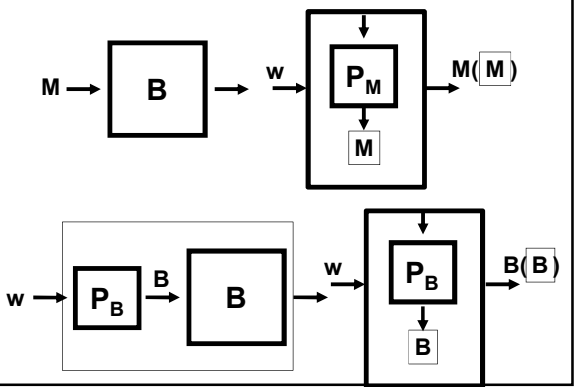
We'll allow one single-sided page of notes

Self-Reference and the Recursion Theorem

Theorem: There is a computable function $q : \Sigma^* \rightarrow \Sigma^*$, where for any string w, q(w) is the *description* of a TM P_w that on any input, prints out w and then accepts



Self-Printing Turing Machine



Another Way of Looking At It

Suppose in general we want to design a program that prints its own description. **How?**

"Print this sentence."

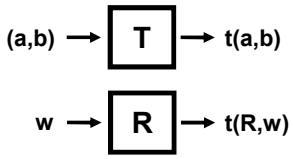
Print two copies of the following, the second copy in quotes: **= B**

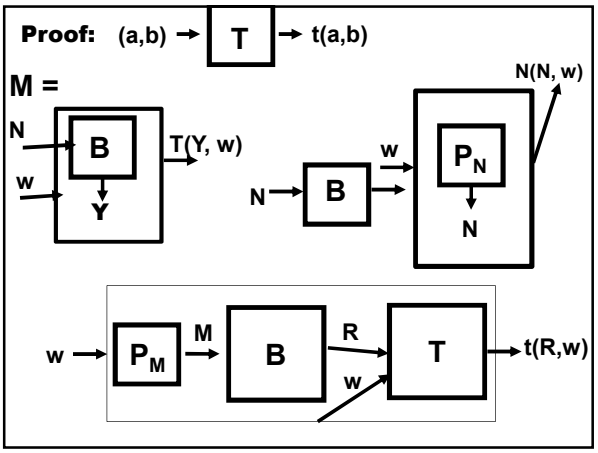
"Print two copies of the following, the second copy in quotes:" **= P_B**

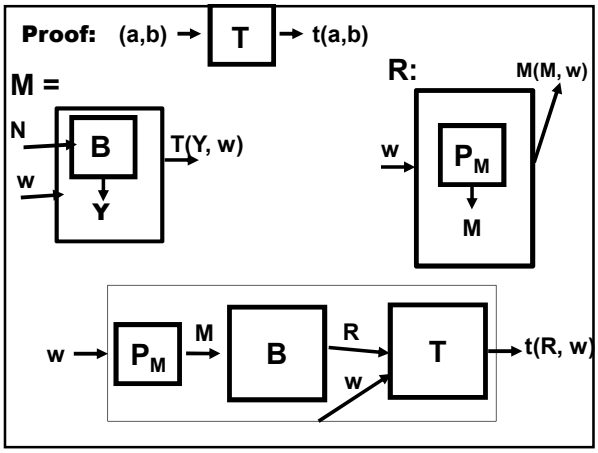
The Recursion Theorem

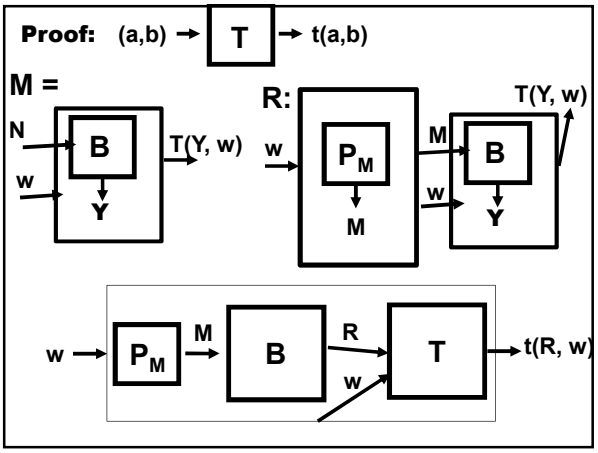
Theorem: Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. There is a Turing machine R that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string w,

$$r(w) = t(R, w)$$









FOO_x(y) := Output x and halt.
BAR(M) := Output "N(x) = Run **FOO_M** outputting M.
Run **M on (M, x)**"
Q(N, x) := Run **BAR**(N) outputting E.
Run **T on (E, x)**

R(x) := Run FOO_Q outputting Q.
Run BAR(Q) outputting D.
Run T on (D, x)

Claim: D is a description of R itself!
D(x) = Run **FOO_Q** outputting **Q.**
Run **Q on (Q, x)**.

FOO_x(y) := Output x and halt.
BAR(M) := Output "N(x) = Run **FOO_M** outputting M.
Run **M on (M, x)**"
Q(N, x) := Run **BAR**(N) outputting E.
Run **T on (E, x)**

R(x) := Run FOO_Q outputting Q.
Run BAR(Q) outputting D.
Run T on (D, x)

Claim: D is a description of R itself!
D(x) = Run **FOO_Q** outputting **Q.**
Run **BAR(Q)** outputting **E.**
Run **T on (E, x)**

Conclusion: D=R, so R(x) = T(R, x)

MORAL:
A Turing machine can obtain its own description, and compute with it

Given any computable t, we can get a computable r such that r(w) = t(R,w) where R is a description of r.

We can use the operation:
"Obtain your own description"
in Turing machine pseudocode!

Theorem: A_{TM} is undecidable

Proof **(using the recursion theorem)**:

Assume H decides A_{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!

The Fixed-Point Theorem

Theorem: Let $t : \Sigma^* \rightarrow \Sigma^*$ be any computable function. There is a TM F such that t(F) describes a TM that is *equivalent* to F.

Proof: Pseudocode for the TM F:

On input w:

1. Obtain the description F
2. Let G be the output of t(F) and interpret G as a TM
3. Accept w iff G(w) accepts

Fixed Points and Rice's Theorem

Rice's Theorem: Let L be a language over TMs that is nontrivial and semantic.
Then L is undecidable.

Proof: Suppose we could decide L.

t(M) :=

If $M \in L$, output a TM M_{NO} s.t. $M_{NO} \notin L$

else output a TM M_{YES} s.t. $M_{YES} \in L$

For all TMs M, the function t(M) always outputs a TM that is *not equivalent* to M.
Contradicts the fixed-point theorem!

Computability in Mathematics

A *formal system* describes a formal language for
- writing mathematical statements,
- has a definition of what statements are "true"
- a definition of a proof of a statement

Example: Any TM M defines some formal system.

- {All mathematical statements} = Σ^*
String w represents the statement "M accepts w"
- {True statements} = L(M)
- Proof that "w is true" =
accepting computation history for M on w

Computability and Mathematics

A formal system $\mathcal{F}$ is *consistent* or *sound* if no false statement has a valid proof in $\mathcal{F}$
(Proof implies Truth)

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

THEOREMS:

For every "interesting" formal system $\mathcal{F}$:

- (Gödel 1931) $\mathcal{F}$ is *incomplete:*
There are true statements that cannot be proved.
- (Gödel 1931) The consistency of $\mathcal{F}$ cannot be proved using proofs in $\mathcal{F}$.
- (Church-Turing 1936) The problem of checking whether a given statement in $\mathcal{F}$ has a proof is undecidable.

Interesting Formal Systems

A formal system $\mathcal{F}$ is *interesting* if:

1. Any mathematical statement describable in English can also be described within $\mathcal{F}$.
Given M and w, there is an $S_{M,w}$ in $\mathcal{F}$ such that $S_{M,w}$ is true in $\mathcal{F}$ iff M accepts w.
2. Proofs are convincing: it should be possible to check that a proof of a theorem is correct
Given (S,P), it is decidable if P is a proof of S in $\mathcal{F}$
3. If there is a proof of S that's describable in English, then there's a proof describable in $\mathcal{F}$.
If M accepts w, then there is a proof in $\mathcal{F}$ of $S_{M,w}$

Unprovable Truths in Mathematics

(Gödel 1931) Every consistent $\mathcal{F}$ is *incomplete:* there are true statements that cannot be proved.

Let $S_{M,w}$ in $\mathcal{F}$ be true iff M accepts w

Proof: Define Turing machine G(x):

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

Claim: S' is true, but has no proof in $\mathcal{F}$
S' basically says "There is no proof for me in $\mathcal{F}$"

(Gödel 1931) The consistency of $\mathcal{F}$ cannot be proved within any interesting consistent $\mathcal{F}$

Proof: Suppose we can prove "$\mathcal{F}$ is consistent" in $\mathcal{F}$

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

G accepts ε iff there is a proof of $\neg S_{G,\varepsilon}$ within $\mathcal{F}$

But if there's a proof in $\mathcal{F}$ of "$\mathcal{F}$ is consistent" then there's a proof in $\mathcal{F}$ that $\neg S_{G,\varepsilon}$ is true:

"If $S_{G,\varepsilon}$ is true, then there is a proof in $\mathcal{F}$ of $\neg S_{G,\varepsilon}$.
But $\mathcal{F}$ 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 is a contradiction.

Undecidability in Mathematics

(Church-Turing 1936) For every interesting $\mathcal{F}$
$PROVABLE_{\mathcal{F}}$ = {S | there's a proof in $\mathcal{F}$ of S, or there's a proof in $\mathcal{F}$ of $\neg$S}
is undecidable.

Proof: Suppose $PROVABLE_{\mathcal{F}}$ is decidable with P.
Then we can decide A_{TM} as follows:
On input (M, w), run the TM P on input $S_{M,w}$
If P accepts, go through all possible proofs in $\mathcal{F}$
If you find a proof of $S_{M,w}$ then accept
If you find a proof of $\neg S_{M,w}$ then reject
If P rejects, then reject.

Next Episode:

Your Midterm…

Good Luck!