# CS 154

## Oracles, Self-Reference, and the Foundations of Mathematics

# Next Tuesday (2/17)

## Midterm: 12:50pm, Bishop Aud

We'll allow one single-sided page of notes

Midterm will cover everything up to and including Tuesday's lecture

If you are an SCPD student, contact SCPD for details about how you will receive your exam

# Rice's Theorem

**Suppose L is a language that satisfies two conditions:**

1. **(Nontrivial)** There are TMs $M_{YES}$ and $M_{NO}$, where $M_{YES} \in L$ and $M_{NO} \notin L$
2. **(Semantic)** For all TMs $M_1$ and $M_2$ such that $L(M_1) = L(M_2)$, $M_1 \in L$ if and only if $M_2 \in L$

**Then, L is undecidable.**

**A Huge Hammer for Undecidability!**

# The Regularity Problem for Turing Machines

$\text{REGULAR}_{\text{TM}}$ = { M | M is a TM and L(M) is regular}

*Given a program, is it equivalent to some DFA?*

**Theorem:** $\text{REGULAR}_{\text{TM}}$ is *not recognizable*

**Proof:** Use Rice's Theorem!

$\text{REGULAR}_{\text{TM}}$ is nontrivial:

- there's an $M_{\emptyset}$ which never halts: $M_{\emptyset} \in \text{REGULAR}_{\text{TM}}$
- there's M' deciding $\{0^n1^n \mid n \geq 0\}$: $M' \notin \text{REGULAR}_{\text{TM}}$

$\text{REGULAR}_{\text{TM}}$ is semantic:

If L(M) = L(M') then L(M) is regular iff L(M') is regular,
therefore $M \in \text{REGULAR}_{\text{TM}}$ iff $M' \in \text{REGULAR}_{\text{TM}}$
By Rice, we have $\neg A_{\text{TM}} \leq_m \text{REGULAR}_{\text{TM}}$

# Oracle Turing Machines and Hierarchies of Undecidable Problems

# Oracle Turing Machines

$q_{YES}$

Is (M, w) in $A_{TM}$?

yes!


| A | N | P | U | T | | | | | |
|---|---|---|---|---|---|---|---|---|---|

INFINITE TAPE

# Oracle Turing Machines

An **oracle Turing machine M** is equipped with a set $B \subseteq \Gamma^*$ to which a TM M may ask membership queries on a special "oracle tape"
[Formally, M enters a special state $q_?$]

and the TM receives a query answer in one step
[Formally, the transition function on $q_?$ is defined in terms of the *entire oracle tape*:
if the string $y$ written on the oracle tape is in $B$,
then state $q_?$ is changed to $q_{YES}$, otherwise $q_{NO}$]

This notion makes sense even if B is not decidable!

**Definition: A is recognizable with B**
**if there is an *oracle TM M with oracle B* that recognizes A**

**Definition: A is decidable with B**
**if there is an *oracle TM M with oracle B* that decides A**

**Language A "Turing-Reduces" to B**

$$A \leq_T B$$

## $A_{TM}$ is decidable with $HALT_{TM}$ ($A_{TM} \leq_T HALT_{TM}$)

On input (M,w), decide if M accepts w as follows:

If (M,w) is in $HALT_{TM}$ then

run M(w) and output its answer.

else REJECT.

## $HALT_{TM}$ is decidable with $A_{TM}$ ($HALT_{TM} \leq_T A_{TM}$)

On input (M,w), decide if M halts on w as follows:

1. If (M,w) is in $A_{TM}$ then ACCEPT

2. Else, switch the accept and reject states of M to get a machine M′. If (M',w) is in $A_{TM}$ then ACCEPT

3. REJECT

# $\leq_T$ versus $\leq_m$

**Theorem:** If $A \leq_m B$ then $A \leq_T B$

**Proof (Sketch):**

If $A \leq_m B$ then there is a computable function
$f : \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

We can simply use one "oracle call" to B to decide A

**Theorem:** $\neg HALT_{TM} \leq_T HALT_{TM}$

**Theorem:** $\neg HALT_{TM} \not\leq_m HALT_{TM}$

***Why?***

# Limitations on Oracle TMs

**The following problem cannot be decided by a TM with an oracle for the Halting Problem:**

**SUPERHALT = { (M,x) | M, with an oracle for the Halting Problem, halts on x}**

***Can still use the diagonalization argument here!***

**Suppose H decides SUPERHALT (with HALT oracle)**

**Define D(X) := "if H(X,X) accepts (with HALT oracle) then LOOP, else ACCEPT."**

**Then D(D) halts ⇔ H(D,D) accepts ⇔ D(D) loops…**

# Limits on Oracle TMs

"Theorem" There is an infinite hierarchy of unsolvable problems!

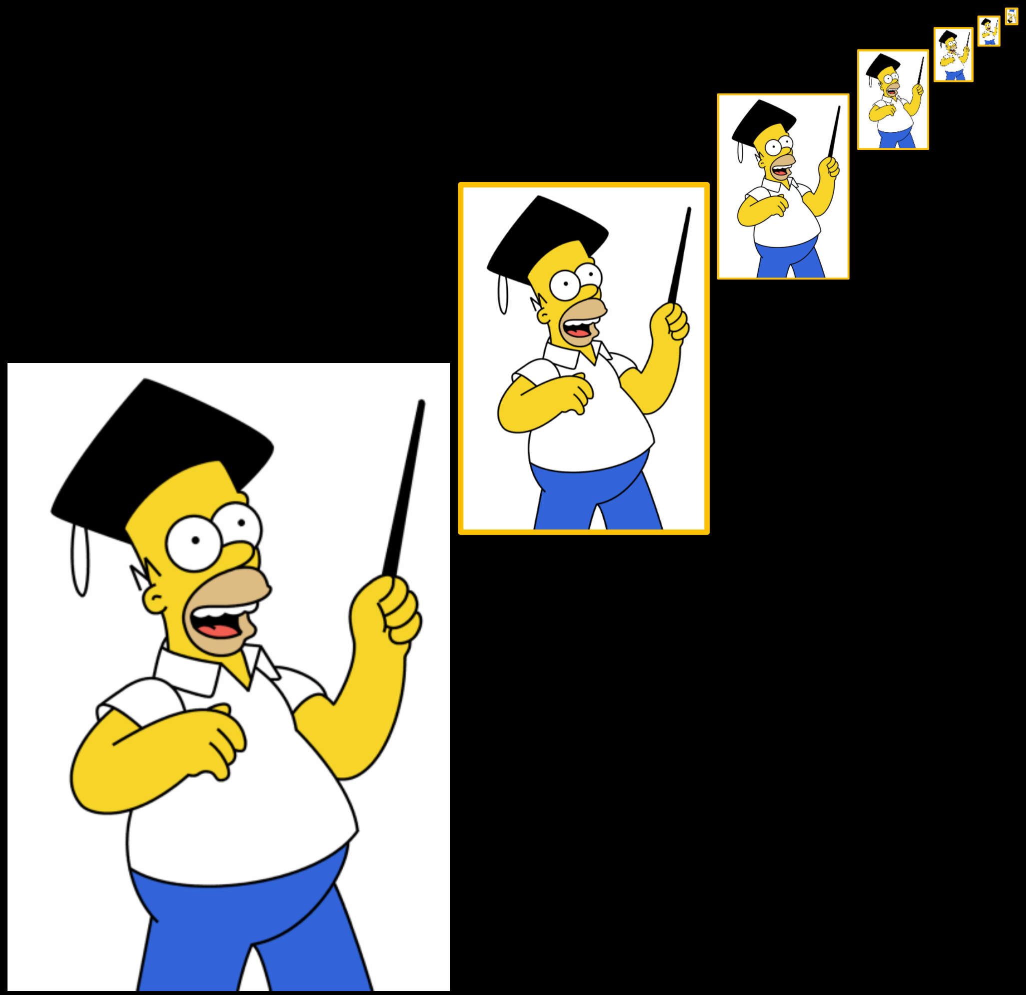
*Given ANY oracle O, there is always a harder problem that can't be decided with that oracle O*

$SUPERHALT^0$ = HALT = { (M,x) | M halts on x}.

$SUPERHALT^1$ = { (M,x) | M, with an oracle for $HALT_{TM}$, halts on x}

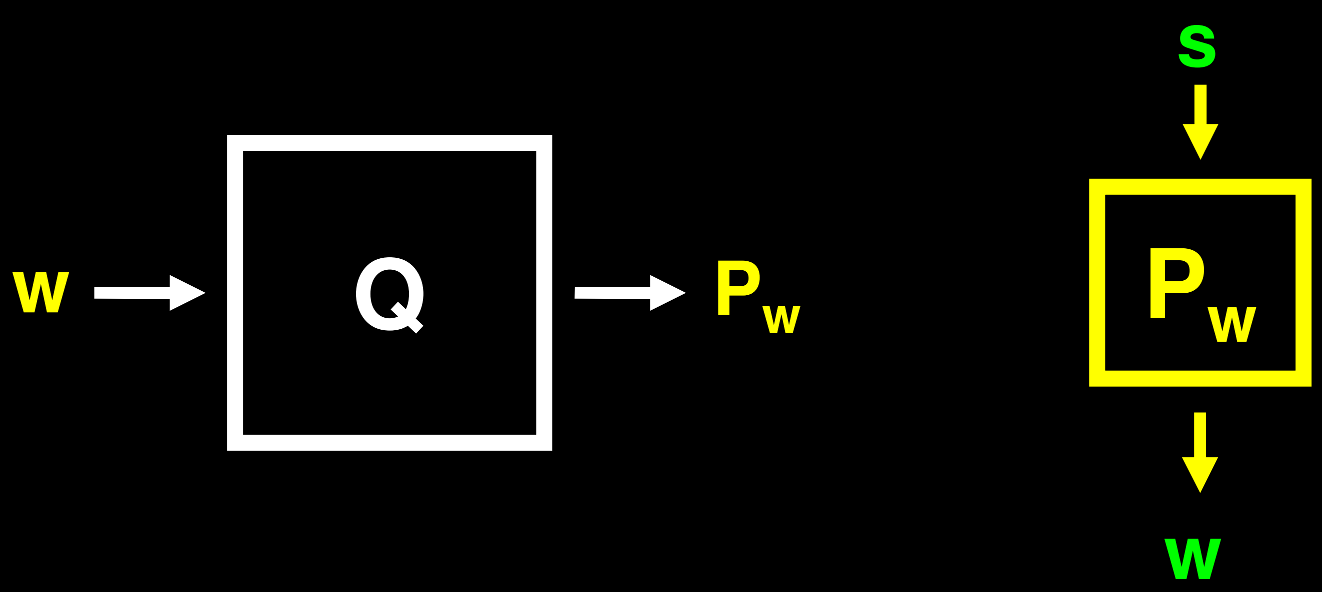
$SUPERHALT^n$ = { (M,x) | M, with an oracle for $SUPERHALT^{n-1}$, halts on x}

# Self-Reference and the Recursion Theorem



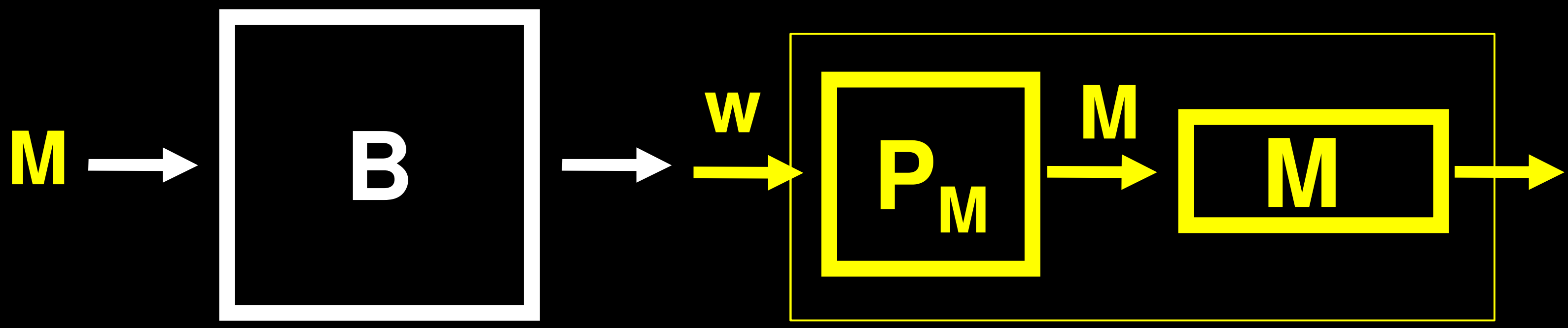
**Lemma:** There is a computable function $q : \Sigma^* \rightarrow \Sigma^*$ such that for any string w, $q(w)$ is the *description* of a TM $P_w$ that on every input, prints out w and then accepts

**"Proof"** Define a TM Q:

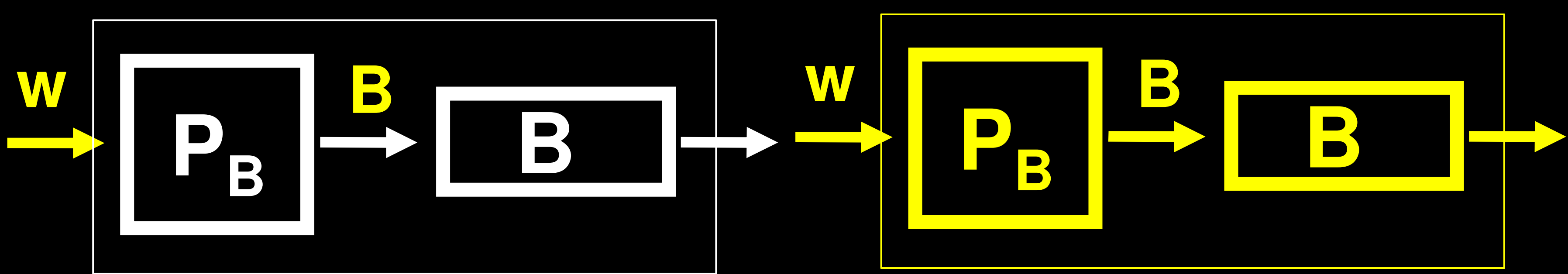


## **Theorem:** There is a Self-Printing TM

**Proof:** First define a TM B:




Now consider the TM:




QED

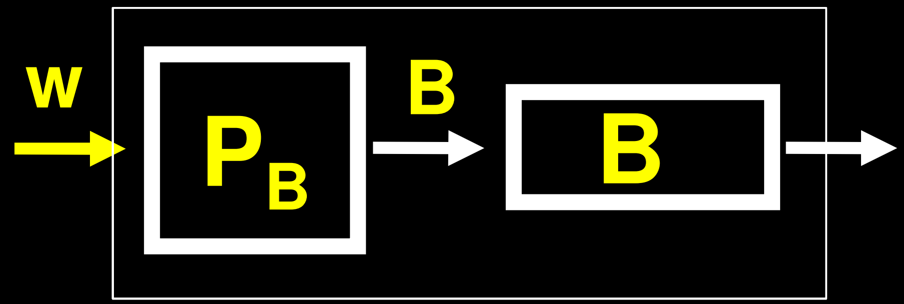
# Another Way of Looking At It

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:" $= P_B$



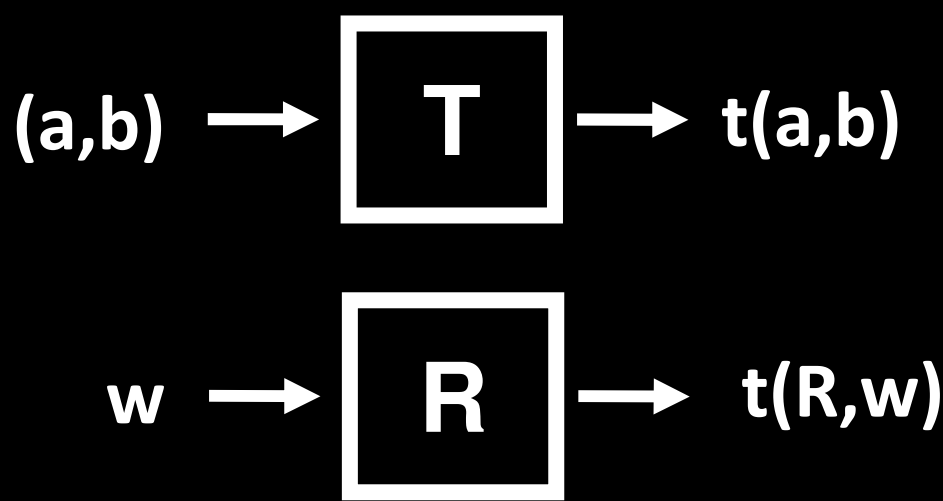
# The Recursion Theorem

**Theorem:** For every TM T computing a function

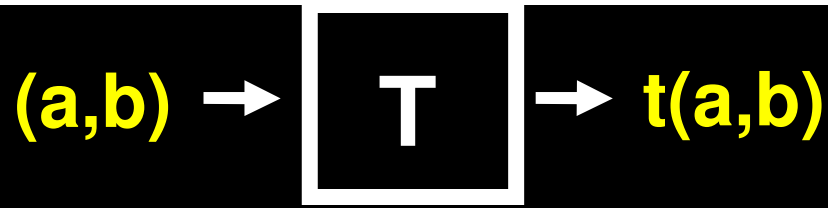
$$t : \Sigma^* \times \Sigma^* \to \Sigma^*$$

there is a Turing machine R computing a function $r : \Sigma^* \to \Sigma^*$, such that for every string w,

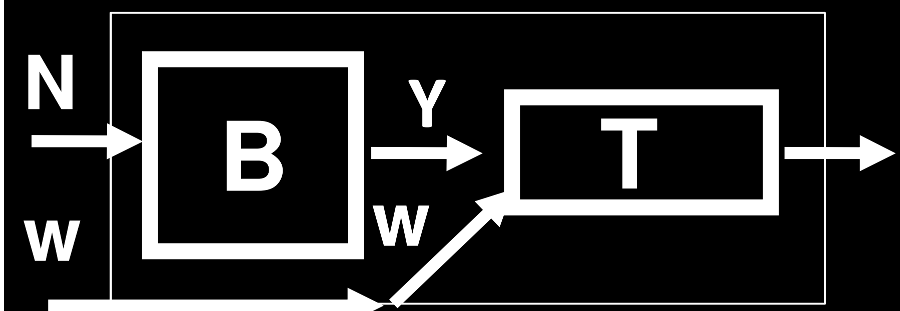
$$r(w) = t(R, w)$$

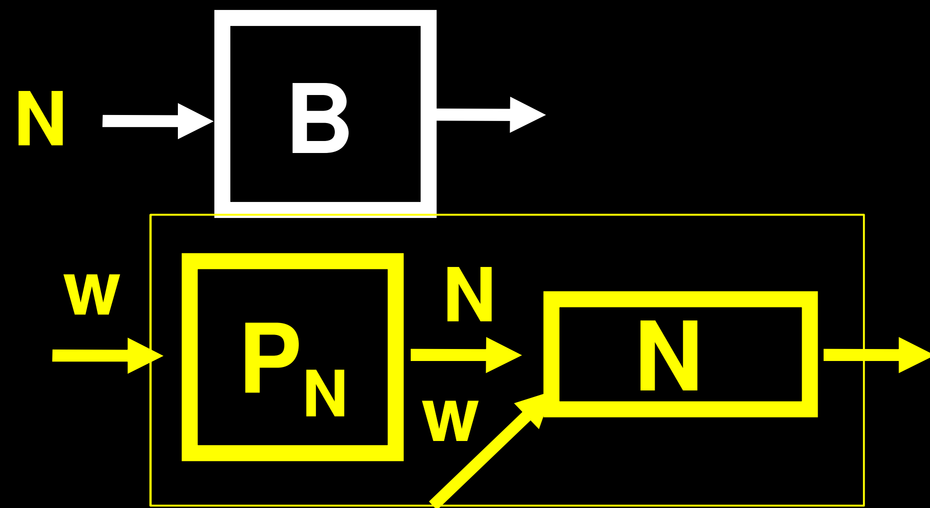


Proof:

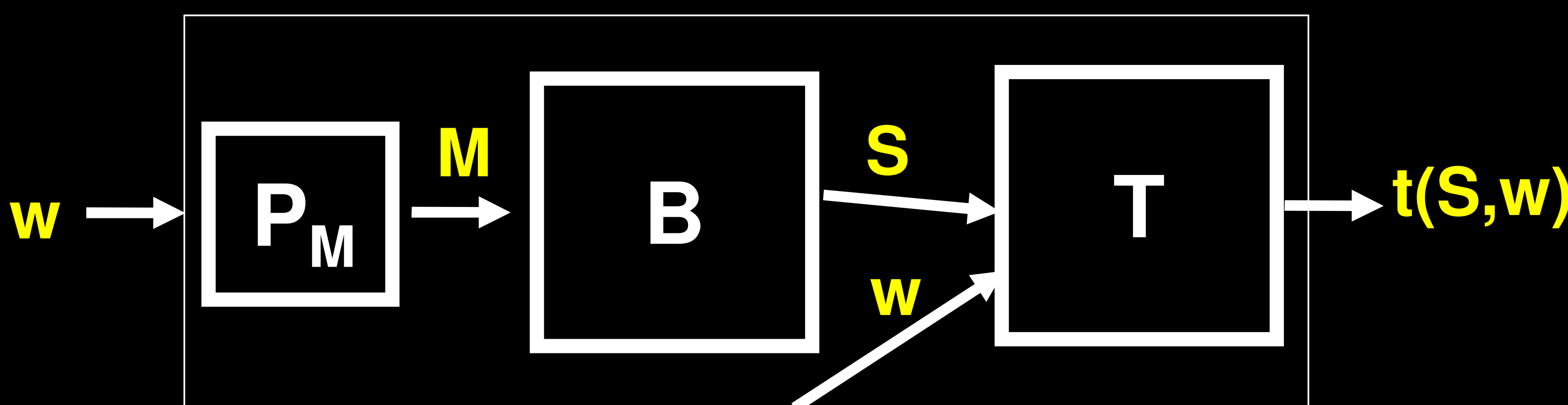



Define M =

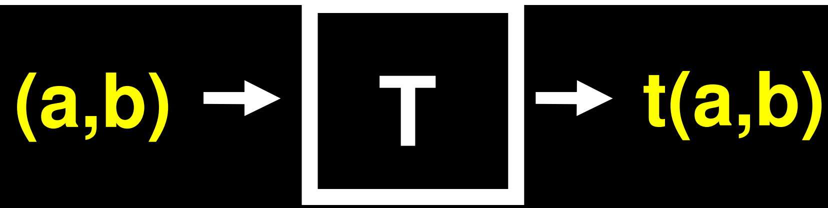


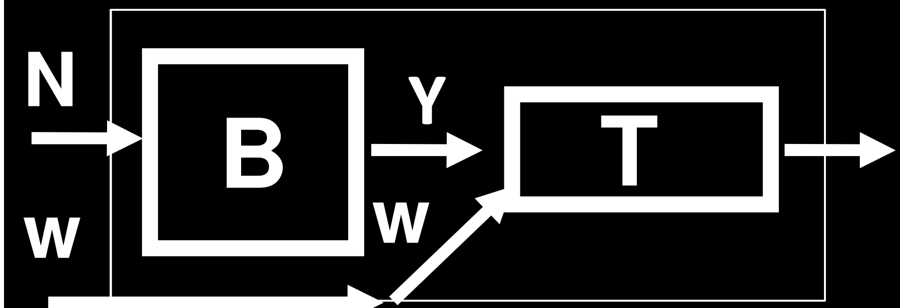

Define R:

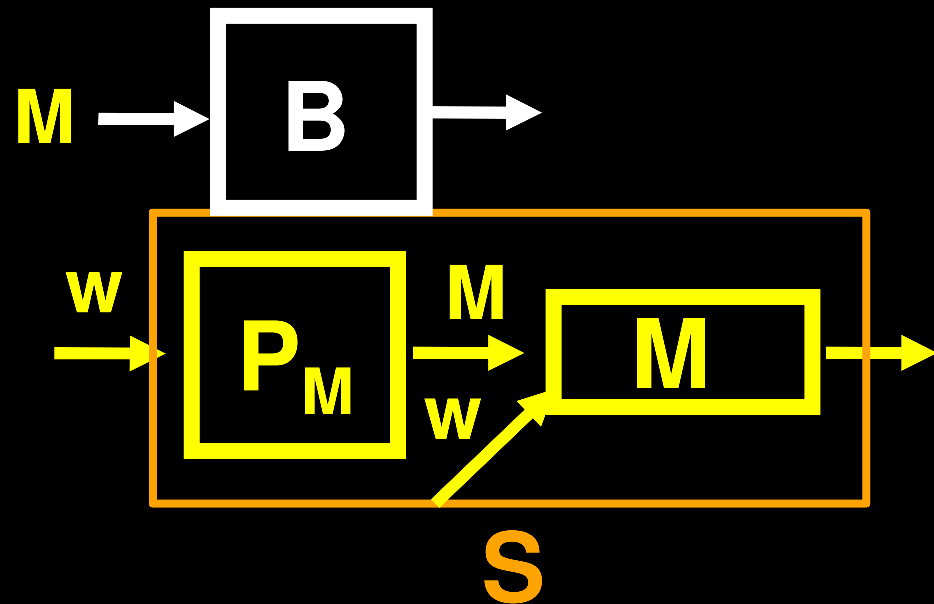


Proof:

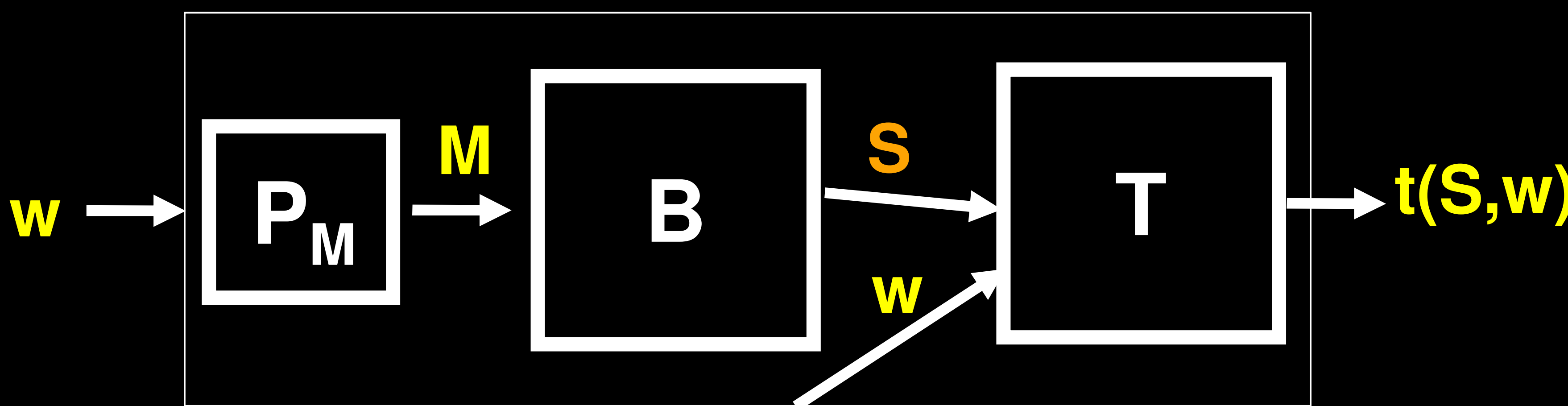



Define M =

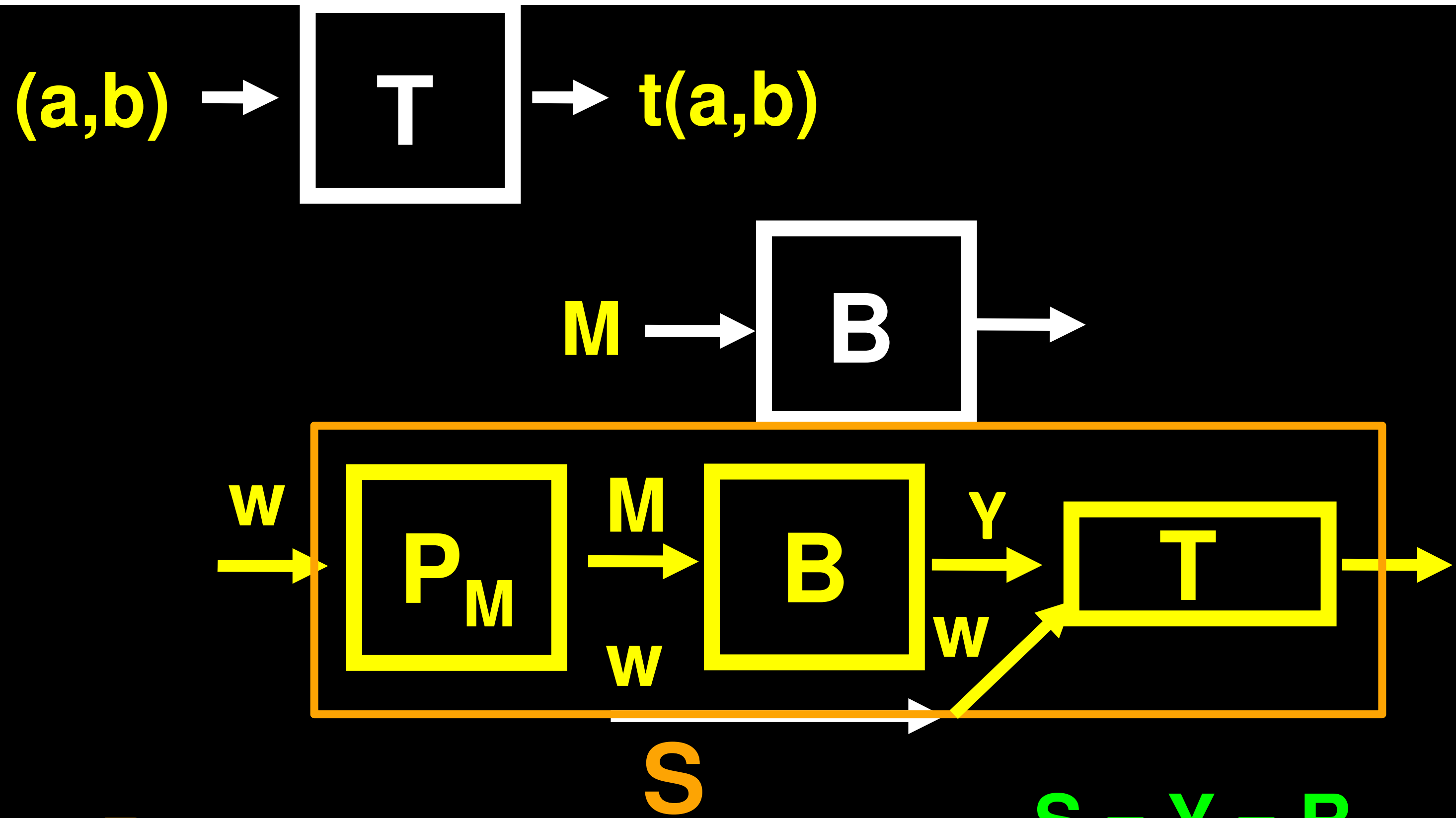



Define R:

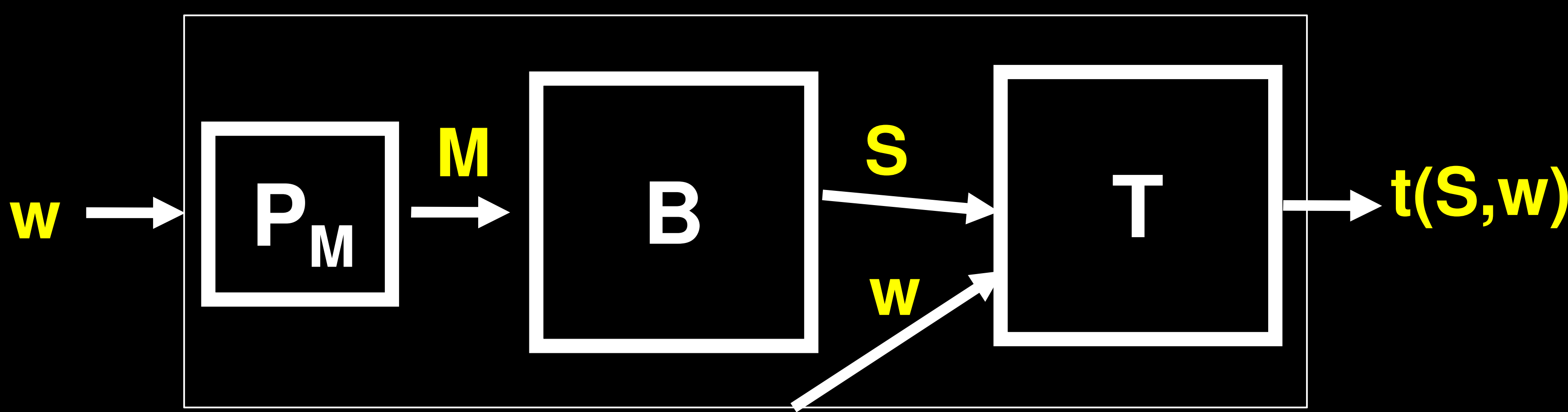


**Proof:**




**S = Y = R**

**Define R:**



**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 bla with x,***

***do bla bla bla with w, etc. etc."***

**We can then find a TM R with the *behavior*:**

***"On input w, do bla bla bla with the description R,***

***do bla bla bla with w, etc. etc."***

**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!

Reminiscent of "free will" paradoxes!

# The Fixed-Point Theorem

**Theorem:** Let $t : \Sigma^* \rightarrow \Sigma^*$ be computable. There is a TM F such that t(F) outputs the description of a TM G such that L(F)=L(G).

**Proof:** Here is pseudocode for the TM F:

On input w:

1. Obtain the description of F
2. Run t(F) and get an output string G.
   Interpret G as the description of a TM
3. Accept w if and only if G accepts w

# Computability and the Foundations of Mathematics

# 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 $\mathcal{F}$

- {Mathematical statements in $\mathcal{F}$} = $\Sigma^*$
  String w represents the statement "M accepts w"
- {True statements in $\mathcal{F}$} = L(M)
- A proof that "M accepts w" can be defined to be
  an accepting computation history for M on w

# Consistency and Completeness

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

**(Proof in $\mathcal{F}$ implies Truth in $\mathcal{F}$)**

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

**(Truth in $\mathcal{F}$ implies Proof in $\mathcal{F}$)**

# Interesting Formal Systems

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

1. **Any mathematical statement about computation can be described as a statement of $\mathcal{F}$.**
   ***Given (M, w), there is an $S_{M,w}$ 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) | P 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 $S_{M,w}$***

# Limitations on Mathematics

**For every consistent and interesting $\mathcal{F}$,**

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

**Theorem 2. (Gödel 1931) The consistency of $\mathcal{F}$ cannot be proved in $\mathcal{F}$.**

**Theorem 3. (Church-Turing 1936) The problem of checking whether a given statement in $\mathcal{F}$ has a proof is undecidable.**

# Unprovable Truths in Mathematics

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

Let $S_{M, w}$ in $\mathcal{F}$ be true if and only if M accepts w

**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 $\mathcal{F}$ over all finite length strings. *Accept* if a proof is found.

**Claim: S' is *true in* $\mathcal{F}$, but has no proof in $\mathcal{F}$**

S' basically says "There is no proof of S' 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 does not accept ε $\Leftrightarrow$ There is no proof of $\neg S_{G,\, \varepsilon}$ in $\mathcal{F}$**

**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 (here's the proof):**

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

PROVABLE$_{\mathcal{F}}$ = {S | there's a proof in $\mathcal{F}$ of S, or there's a proof in $\mathcal{F}$ of ¬S}

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

**Proof:** Suppose PROVABLE$_{\mathcal{F}}$ 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 $\mathcal{F}$

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?

# Next Episode:

**Your Midterm… Good Luck!**