6.045

Lecture 12:

Deep Computability

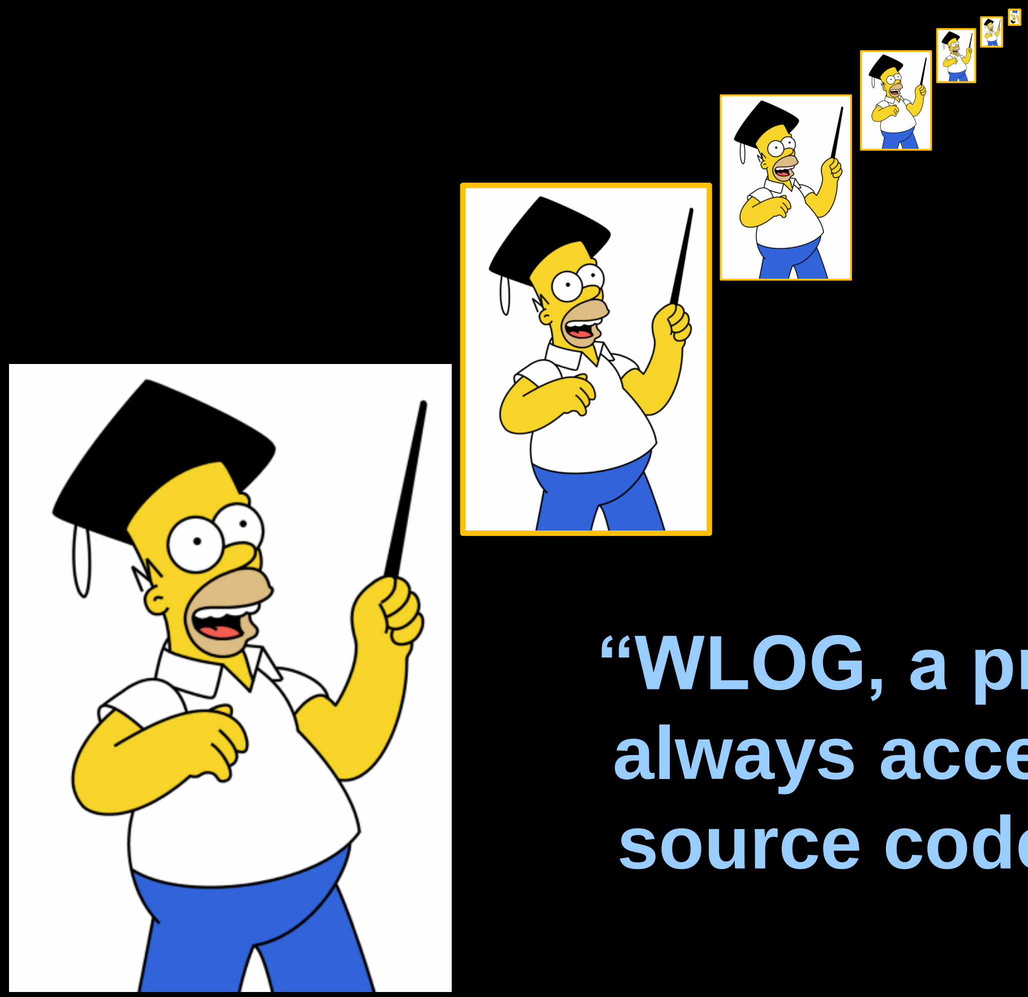
Self-Reference in Computation and The Foundations of Mathematics

6.045

Announcements:

- Midterm exam still set for April 2
  - Will cover Lectures 1-11
- Today's material is not on the midterm

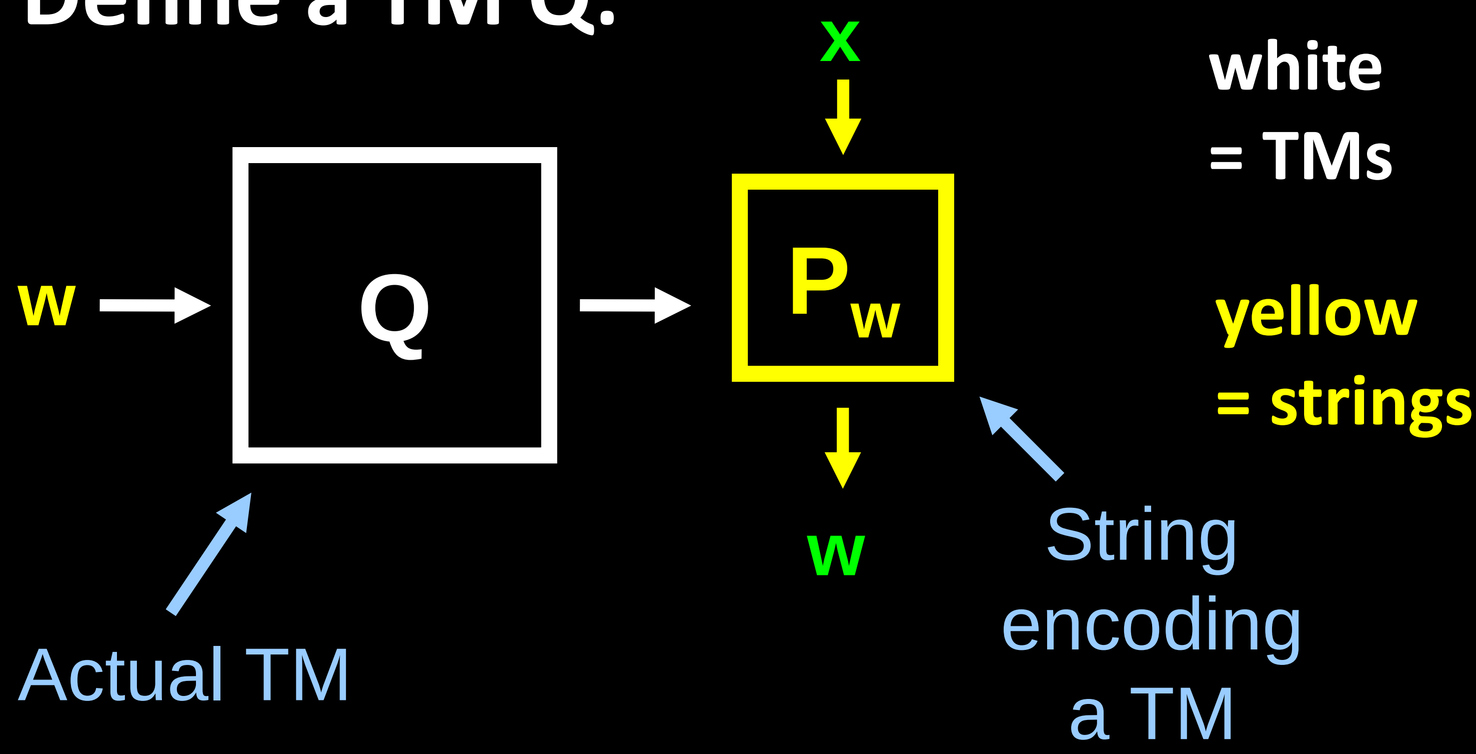
Self-Reference and the Recursion Theorem



“WLOG, a program can always access its own source code as input”

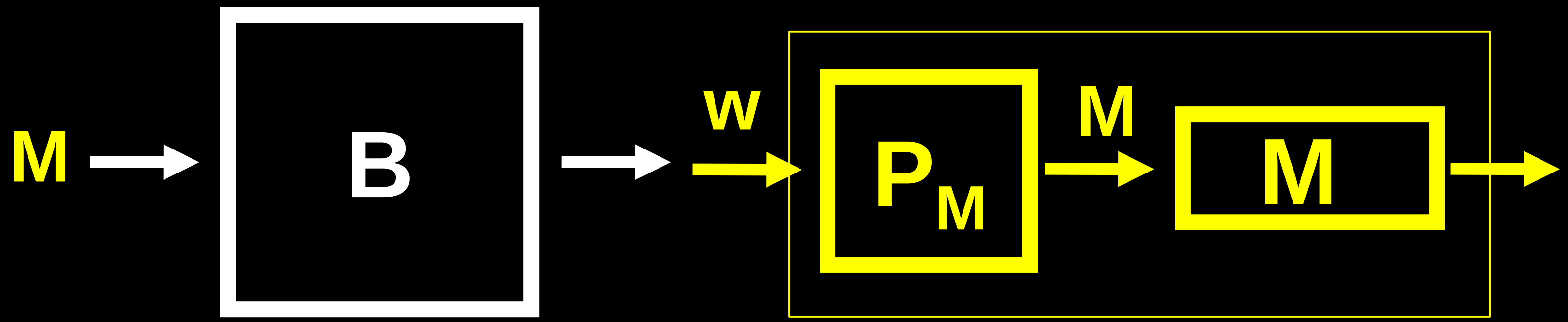
Lemma: There is a computable function $q : \Sigma^* \rightarrow \Sigma^*$ such that for every string w, q(w) is the *description* of a TM P_w that on every input, prints out w and then accepts

"Proof" Define a TM Q:

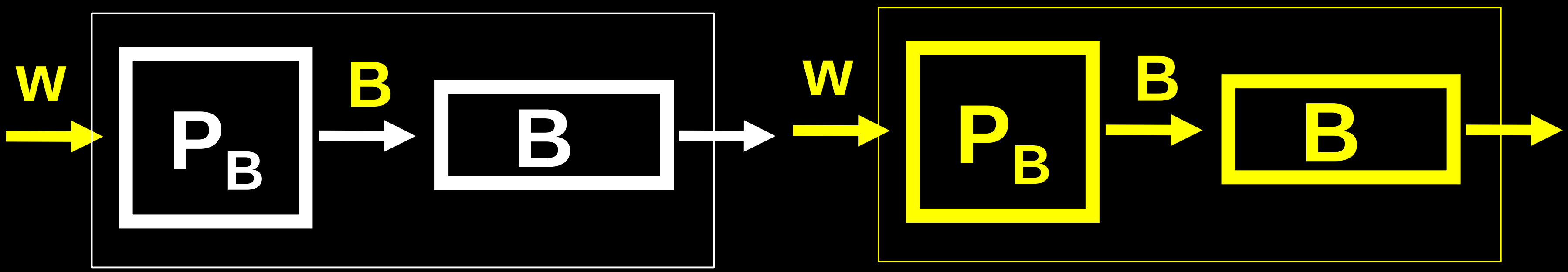


Theorem: There is a Self-Printing TM

Proof: First define a TM B which does this:



Now consider the TM that looks like this:



This is a TM that prints its own description! QED

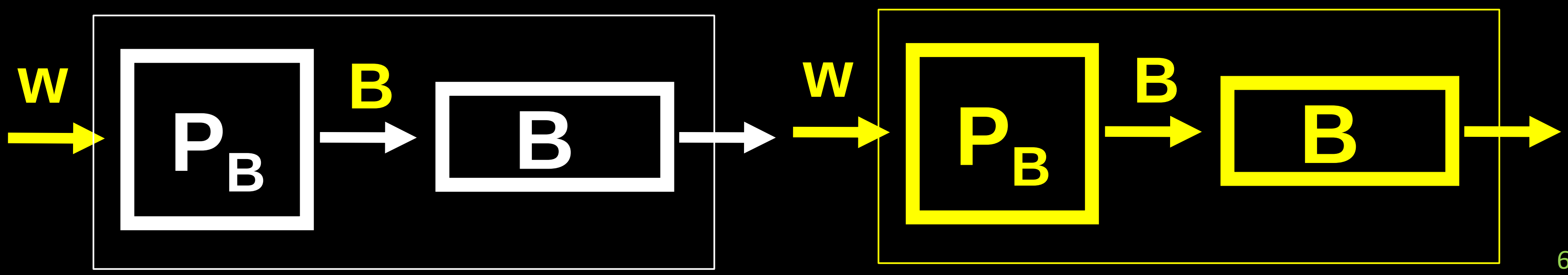
Another Way of Looking At It

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: $\approx$ B

"Print two copies of the following, the second copy in quotes:" $\approx P_B$



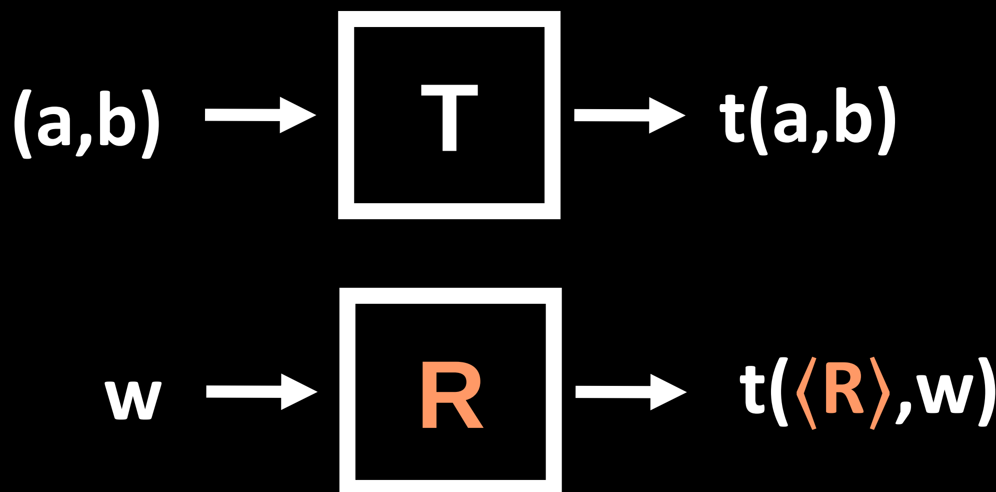
The Recursion Theorem

Theorem: For every TM T computing a function

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

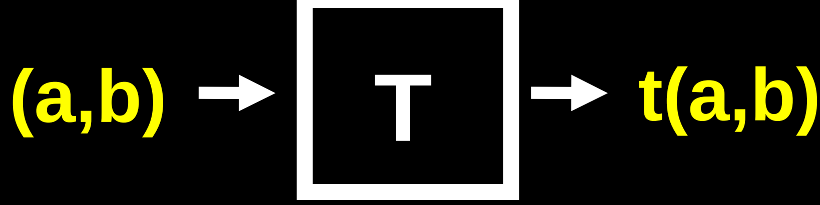
there is a Turing machine R computing a function $R : \Sigma^* \to \Sigma^*$, such that for every string w,

$$R(w) = t(\langle R \rangle, w)$$

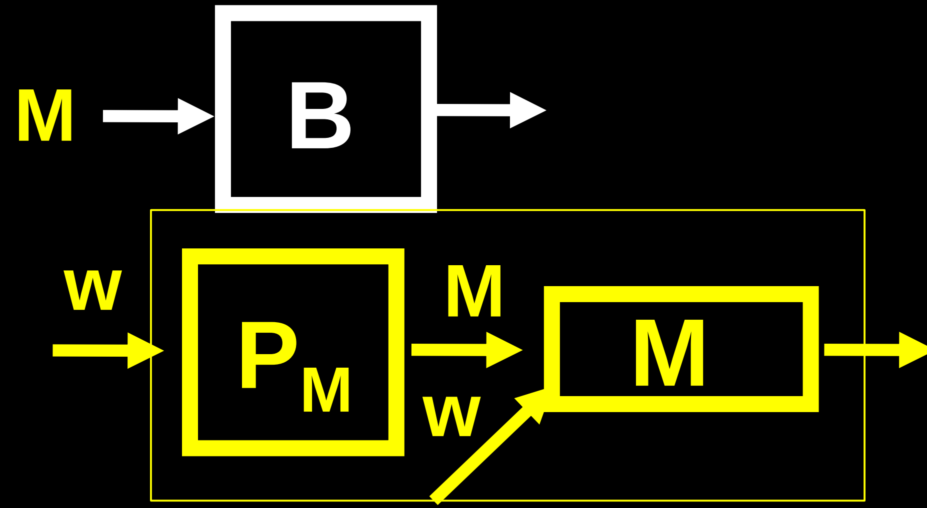


"TMs can implement recursion!"

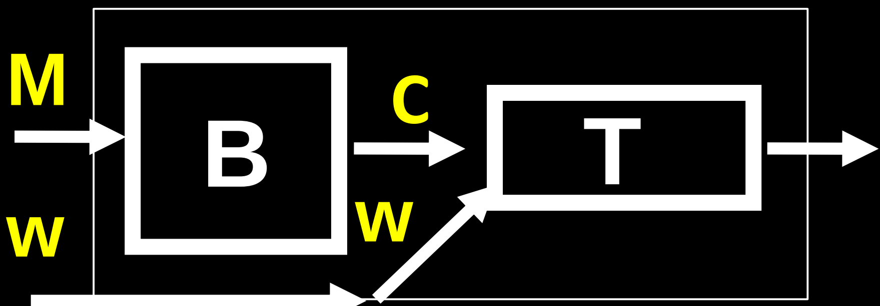
Proof:



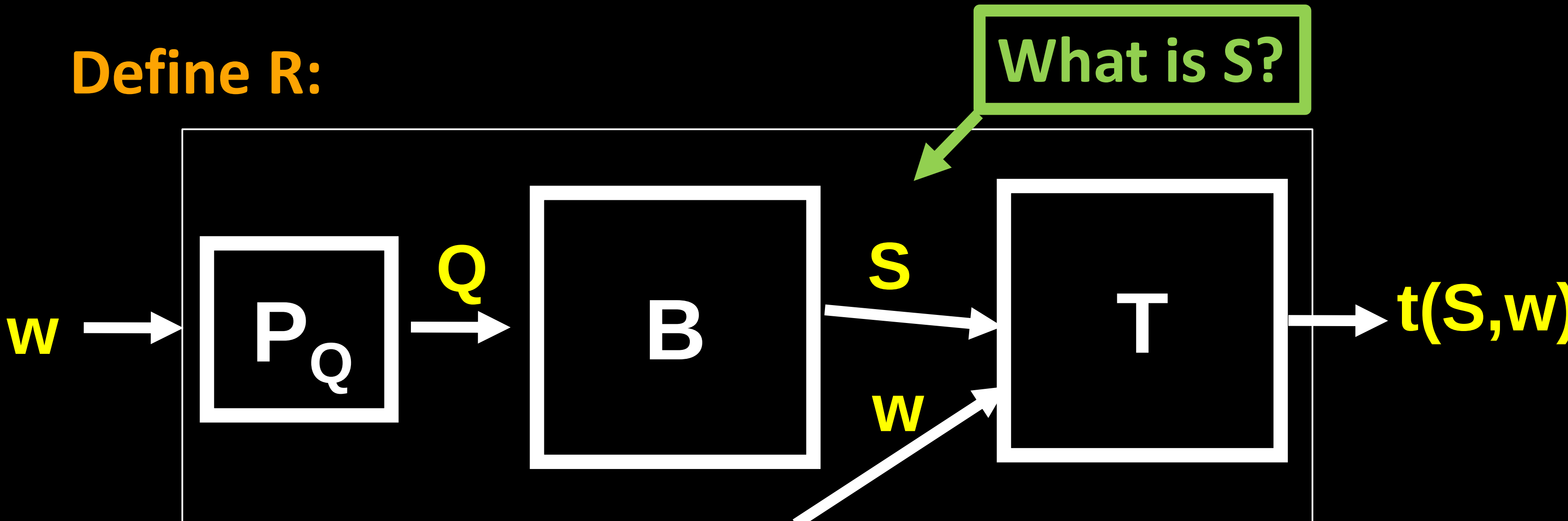
Define TM B:



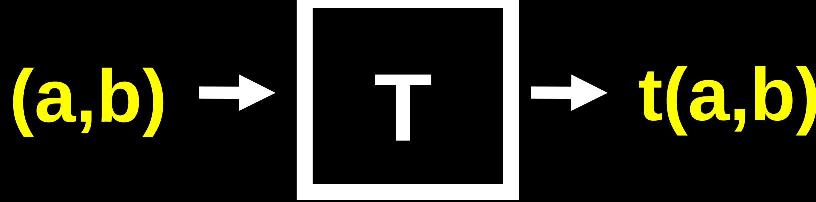
Define TM Q:



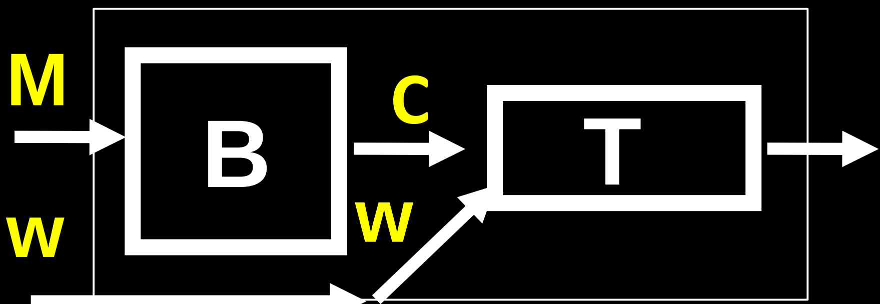
Define R:

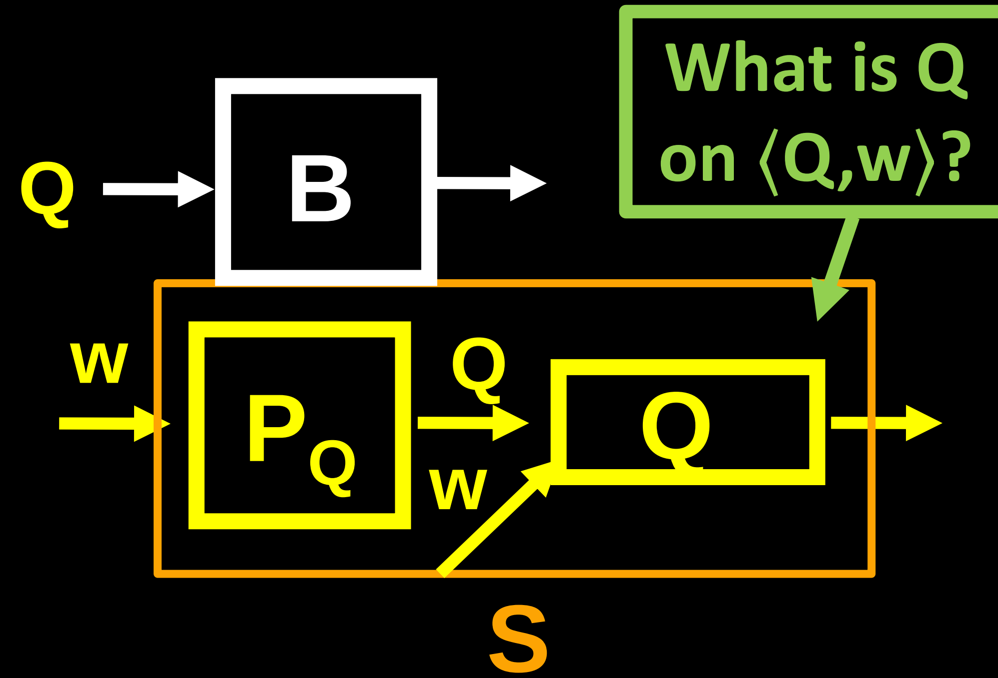


Proof:

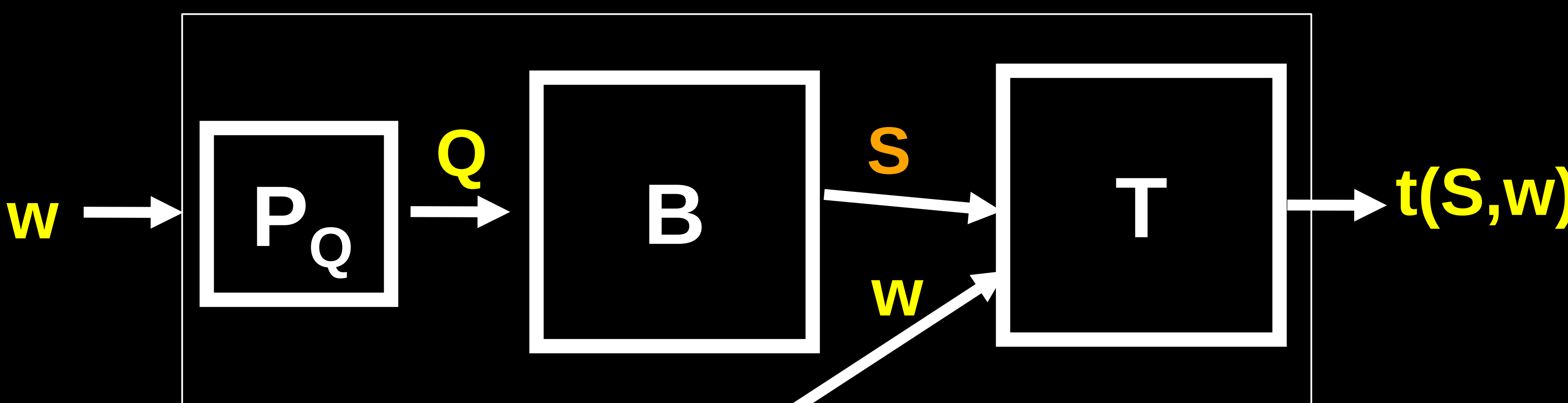


Define TM Q:

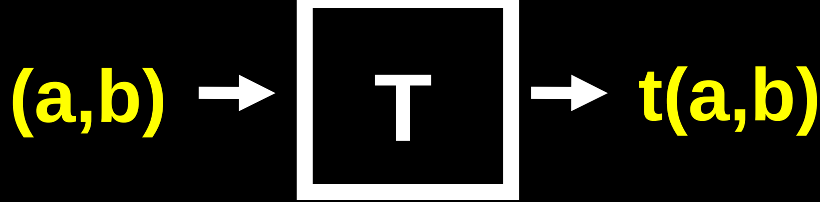


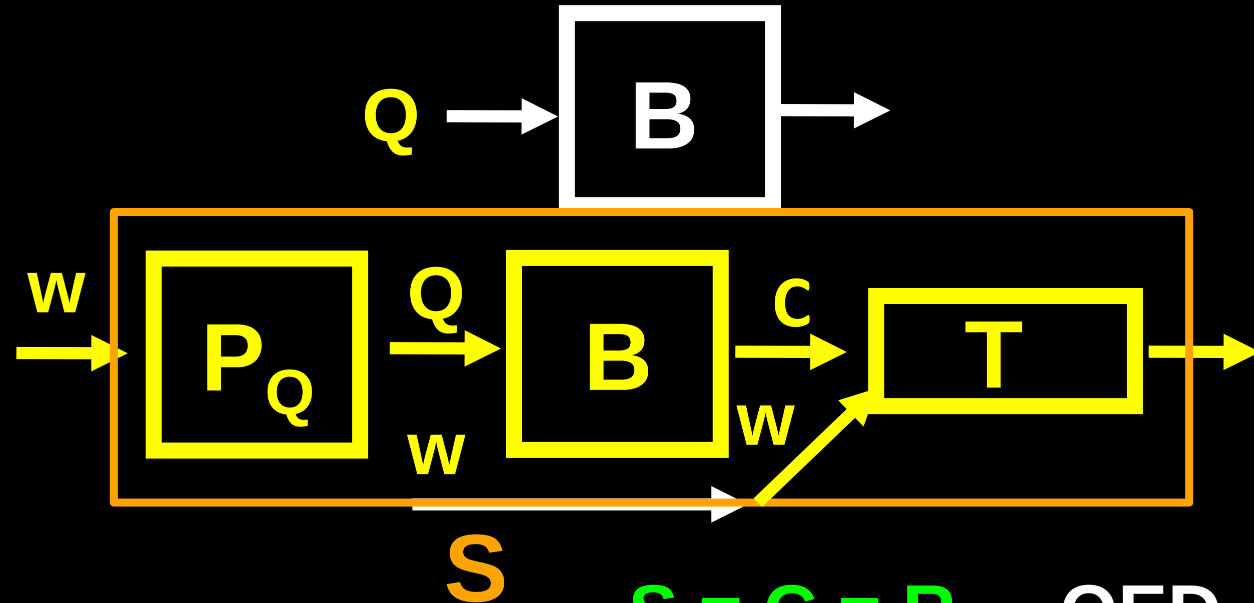


Define R:



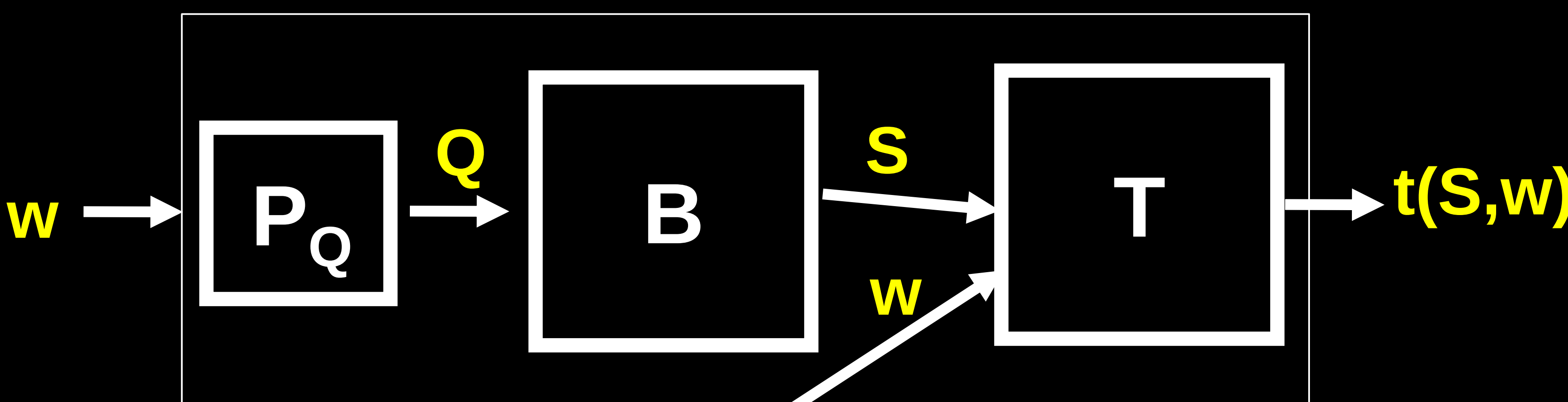
Proof:





S = C = R. **QED**

Define R:



$FOO_x(y)$:= Output x and halt.

$BAR(\langle M \rangle)$:= Output "$N(w)$ = Run $FOO_{<M>}$ outputting $\langle M \rangle$.
Run M on $(\langle M \rangle, w)$"

$Q(\langle M \rangle, w)$:= Run $BAR(\langle M \rangle)$ outputting $\langle N \rangle$.
Run T on $(\langle N \rangle, w)$

$R(w)$:= Run $FOO_{<Q>}$ outputting $\langle Q \rangle$.
Run $BAR(\langle Q \rangle)$ outputting $\langle N \rangle$.
Run T on $(\langle N \rangle, x)$

Claim: $\langle N \rangle$ is a description of R itself!

$N(w)$ = Run $FOO_{<Q>}$ outputting $\langle Q \rangle$.
Run Q on $(\langle Q \rangle, w)$

$FOO_x(y)$:= Output x and halt.

$BAR(\langle M \rangle)$:= Output "$N(w)$ = Run $FOO_{<M>}$ outputting $\langle M \rangle$.
Run M on $(\langle M \rangle, w)$"

$Q(\langle M \rangle, w)$:= Run $BAR(\langle M \rangle)$ outputting $\langle N \rangle$.
Run T on $(\langle N \rangle, w)$

$R(w)$:= Run $FOO_{<Q>}$ outputting $\langle Q \rangle$.
Run $BAR(\langle Q \rangle)$ outputting $\langle N \rangle$.
Run T on $(\langle N \rangle, w)$

Claim: $\langle N \rangle$ is a description of R itself!

$N(w)$ = Run $FOO_{<Q>}$ outputting $\langle Q \rangle$.
Run $BAR(\langle Q \rangle)$ outputting $\langle N \rangle$.
Run T on $(\langle N \rangle, w)$

Therefore $R(w) = T(\langle R \rangle, w)$

For every computable t, there is a computable r such that $r(w) = t(\langle R \rangle, w)$ where *R is a Turing machine computing r*

Moral: **Suppose we can design a TM T of the form**

"On input (x,w), do bla bla with x,
do bla bla bla with w, etc. etc."

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

"On input w, do bla bla with the code of R,
do bla bla bla with w, etc. etc."

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

Theorem: $A_{TM} = \{\langle M,w \rangle \mid M \text{ accepts } w\}$ is undecidable

Proof (using the recursion theorem)

Assume H decides A_{TM} Define a TM T as follows:

$T(\langle M \rangle, w)$:= Run H on $\langle M,w \rangle$. If H accepts, then reject.
If H rejects, then accept.

Recursion Theorem $\Rightarrow$ $\exists$ TM B such that for all w,
$B(w) = T(\langle B \rangle, w)$.

Now, running B on w outputs $T(\langle B \rangle, w)$, which is the *opposite answer* of H on $\langle B,w \rangle$. Contradiction!

A formalization of "free will" paradoxes!
No single machine can predict behavior of all others

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 any input w always does the *opposite* of what H on ⟨B,w⟩ says B would do! Contradiction!

Turing Machine Minimization

MIN = {⟨M⟩ | M is a minimal-state TM over Γ={0,1,□}}

Theorem: MIN is undecidable

Proof: Suppose we could recognize MIN with TM M'

$M(x)$:= Obtain the description of M.

For k = 1,2,3,…

Run M' on the first k TMs $M_1, \ldots M_k$ for k steps,

Until M' accepts some M_i with *more* states than M

Output M_i on x.

Why does M_i exist?

We have: 1. $L(M) = L(M_i)$ [by construction]

2. M has *fewer* states than M_i

3. M_i is minimal [by definition of MIN]

CONTRADICTION!

Computability, Logic, and the Foundations of Mathematics:

Math is Incomplete!

Formal Systems of Mathematics

A *formal system* describes a formal language for
- writing (finite) mathematical statements as strings,
- has a definition of a proof of a statement (as strings)
- has a notion of "true" statements

Example: Every TM M can be used to define a formal system $\mathcal{F}$ with the properties:

- {Mathematical statements in $\mathcal{F}$} = Σ*
  String w represents the statement "M halts on w"

- A proof of "M halts on w" can be defined as the *computation history* of M on w: the sequence of configurations $C_0\, C_1 \cdots C_t$ that M goes through while computing on w

Interesting Systems of Mathematics

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

1. **Mathematical statements about computation can be (computably) described as a statement of $\mathcal{F}$.**
   Given (M, w), there is a (computable) $S_{M,w}$ of $\mathcal{F}$ 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 candidate proof of a theorem is correct.**
   ***This set is decidable:* {(S, P) | P is a proof of S in $\mathcal{F}$}**
3. **Mathematical proofs with computation histories can be expressed in $\mathcal{F}$.**
   If TM M halts on w, then there's either a proof P of $S_{M,w}$ or a proof P of $\neg S_{M,w}$

Consistency and Completeness

A formal system $\mathcal{F}$ is ***inconsistent*** if
there is a statement S in $\mathcal{F}$
such that both S *and* $\neg S$ are provable in $\mathcal{F}$
$\mathcal{F}$ is **consistent** if it is NOT inconsistent

A formal system $\mathcal{F}$ is ***incomplete*** if
there is a statement S in $\mathcal{F}$
such that neither S *nor* $\neg S$ are provable in $\mathcal{F}$
$\mathcal{F}$ is **complete** if it is NOT incomplete

We want consistent and complete systems!

Limitations on Mathematics!

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

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

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 statements that cannot be proved or disproved.

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

Proof: Define TM G(w):

1. Obtain own description G [Recursion Theorem!]
2. For all strings P in lexicographical order,
   - If (P is a proof of $S_{G,w}$ in $\mathcal{F}$) then reject
   - If (P is a proof of $\neg S_{G,w}$ in $\mathcal{F}$) then accept

Note: If $\mathcal{F}$ is complete then G cannot run forever!

1. If (G accepts w) then have proof P of "G doesn't accept w"
2. If (G rejects w) then have proof P of "G accepts w"

In either case, $\mathcal{F}$ is inconsistent! Proof of $S_{G,w}$ and $\neg S_{G,w}$

Unprovable Truths in Mathematics

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

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

Proof: Define TM G(w):

1. Obtain own description G [Recursion Theorem!]
2. For all strings P in lexicographical order,

    If (P is a proof of $S_{G,w}$ in $\mathcal{F}$) then reject

    If (P is a proof of $\neg S_{G,w}$ in $\mathcal{F}$) then accept

Note: If $\mathcal{F}$ is complete then G cannot run forever!

Conclusion: G must run forever.

So in fact $\neg S_{G,w}$ is a true statement, but it (and its negation) have no proof in $\mathcal{F}$!

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

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

We constructed $\neg S_{G,\,w}$ = "G does not accept w"

which *has no proof* in $\mathcal{F}$

G accepts w ⇒ There are proofs of $S_{G,\,w}$ and $\neg S_{G,\,w}$ in $\mathcal{F}$

But if there's a proof P of "$\mathcal{F}$ is consistent" in $\mathcal{F}$, then there *is* a proof of $\neg S_{G,\,w}$ in $\mathcal{F}$ (here's the proof):

"$\mathcal{F}$ is consistent, because *<insert proof P here>*.
If $S_{G,\,w}$ is true, then both $S_{G,w}$ and $\neg S_{G,\,w}$ have proofs in $\mathcal{F}$.
But $\mathcal{F}$ is consistent, so this is a contradiction.
Therefore, $\neg S_{G,\,w}$ is true."

This contradicts the previous theorem!

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 $\neg$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 could decide A_{TM} with the following procedure:

On input (M, w), run the TM P on input $S_{M,w}$

If P accepts, examine all proofs in lex order
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!