6.045: Automata, Computability, and Complexity
Or, Great Ideas in Theoretical Computer Science
Spring, 2010

Class 10
Nancy Lynch

Today

- Final topic in computability theory: Self-Reference and the Recursion Theorem
- Consider adding to TMs (or programs) a new, powerful capability to “know” and use their own descriptions.
- The Recursion Theorem says that this apparent extra power does not add anything to the basic computability model: these self-referencing machines can be transformed into ordinary non-self-referencing TMs.

Today

- Self-Reference and the Recursion Theorem
- Topics:
  - Self-referencing machines and programs
  - Statement of the Recursion Theorem
  - Applications of the Recursion Theorem
  - Proof of the Recursion Theorem: Special case
  - Proof of the Recursion Theorem: General case
- Reading:
  - Sipser, Section 6.1

Self-referencing machines and programs

Self-referencing machines/programs

- Consider the following program P_1.
- P_1:
  - Obtain $< P_1 >$
  - Output $< P_1 >$
- P_1 simply outputs its own representation, as a string.
- Simplest example of a machine/program that uses its own description.

Self-referencing machines/programs

- A more interesting example:
- P_2: On input w:
  - If $w = \varepsilon$ then output 0
  - Else
    - Obtain < P_2 >
    - Run P_2 on tail(w)
    - If P_2 on tail(w) outputs a number n then output n+1.
- What does P_2 compute?
- It computes |w|, the length of its input.
- Uses the recursive style common in LISP, Scheme, other recursive programming languages.
- We assume that, once we have the representation of a machine, we can simulate it on a given input.
- E.g., if P_2 gets < P_2 >, it can simulate P_2 on any input.

Self-referencing machines/programs

- One more example:
- P_3: On input w:
  - Obtain < P_3 >
  - Run P_3 on w
  - If P_3 on w outputs a number n then output n+1.
- A valid self-referencing program.
- What does P_3 compute?
- Seems contradictory: if P_3 on w outputs n then P_3 on w outputs n+1.
- But according to the usual semantics of recursive calls, it never halts, so there's no contradiction.
- P_3 computes a partial function that isn't defined anywhere.

Statement of the Recursion Theorem

The Recursion Theorem

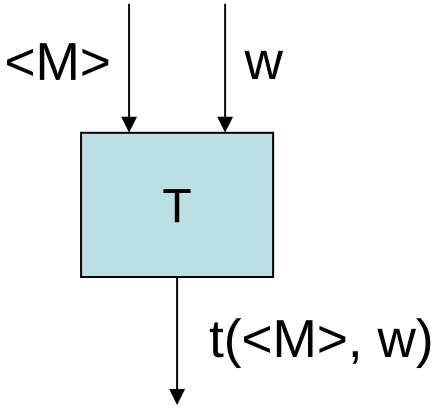
- Used to justify self-referential programs like P_1, P_2, P_3, by asserting that they have corresponding (equivalent) basic TMs.
- Recursion Theorem (Sipser Theorem 6.3):

Let T be a TM that computes a (possibly partial) 2-argument function t: $\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$.

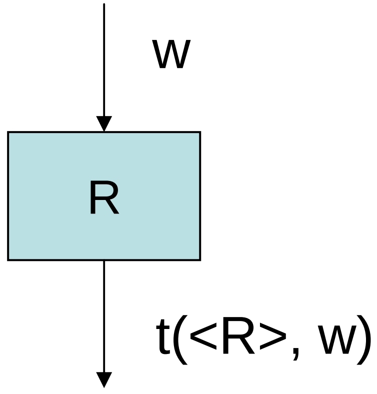
Then there is another TM R that computes the function r: $\Sigma^* \rightarrow \Sigma^*$, where for any w, r(w) = t(<R>, w).

The Recursion Theorem

- Recursion Theorem: Let T be a TM that computes a (possibly partial) 2-argument function t: $\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Then there is another TM R that computes the function r: $\Sigma^* \rightarrow \Sigma^*$, where for any w, r(w) = t(<R>, w).



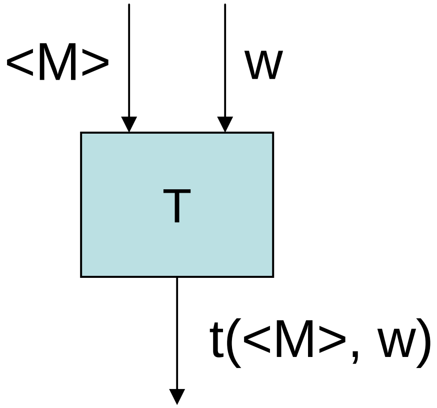
- Thus, T is a TM that takes 2 inputs.
- Think of the first as the description of some arbitrary 1-input TM M.

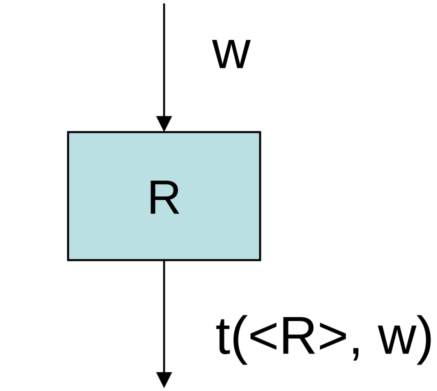


- Then R behaves like T, but with the first input set to <R>, the description of R itself.
- Thus, R uses its own representation.

The Recursion Theorem

- Recursion Theorem: Let T be a TM that computes a (possibly partial) 2-argument function $t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Then there is another TM R that computes the function $r: \Sigma^* \rightarrow \Sigma^*$, where for any w, r(w) = t(<R>, w).

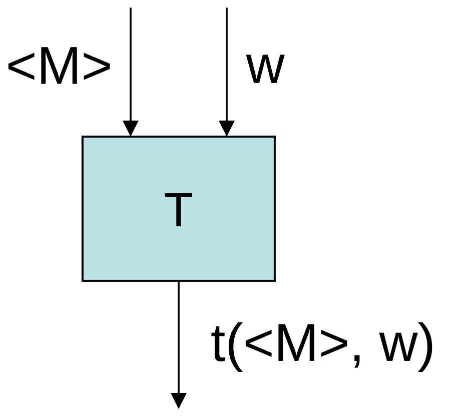


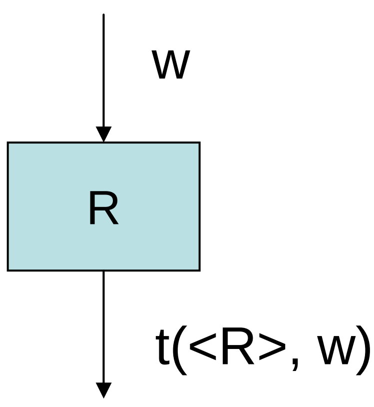


- Example: P_2, revisited
  - Computes length of input.
  - What are T and R?
  - Here is a version of P_2 with an extra input <M>:
  - T_2: On inputs <M> and w:
    - If $w = \varepsilon$ then output 0
    - Else run M on tail(w); if it outputs n then output n+1.

The Recursion Theorem

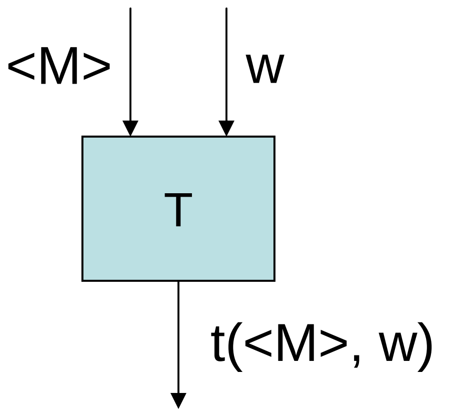
- Example: P_2, revisited
  - T_2: On inputs <M> and w:
    - If w = ε then output 0
    - Else run M on tail(w); if it outputs n then output n+1.
  - T_2 produces different results, depending on what M does.
  - E.g., if M always loops:
    - T_2 outputs 0 on input w = ε and loops on every other input.
  - E.g., if M always halts and outputs 1:
    - T_2 outputs 0 on input w = ε and outputs 2 on every other input.

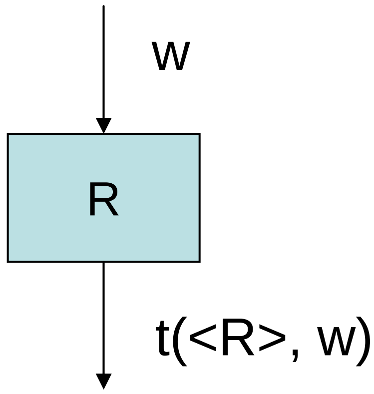




The Recursion Theorem

- Example: P_2, revisited
  - T_2: On inputs <M> and w:
    - If $w = \varepsilon$ then output 0
    - Else run M on tail(w); if it outputs n then output n+1.
  - Recursion Theorem says there is a TM R computing t(<R>, w)---just like T_2 but with input <M> set to <R> for the same R.
  - This R is just P_2 as defined earlier.

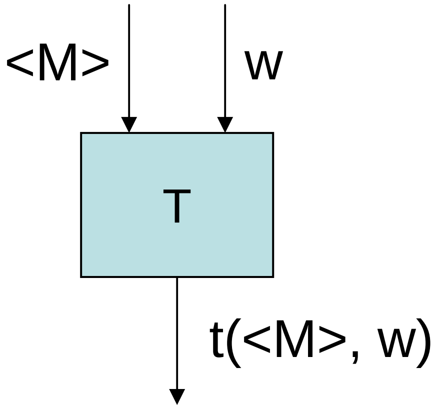




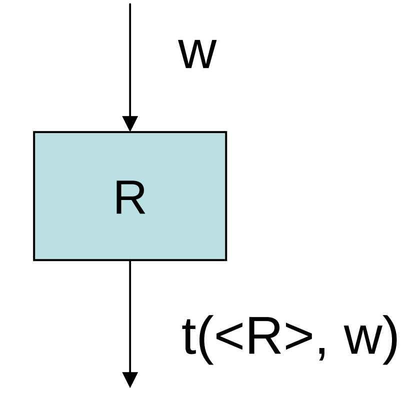
The Recursion Theorem

- Recursion Theorem (Sipser Theorem 6.3):

  Let T be a TM that computes a (possibly partial) 2-argument function t: $\Sigma^* \times \Sigma^* \to \Sigma^*$.



  Then there is another TM R that computes the function r: $\Sigma^* \to \Sigma^*$, where for any w, r(w) = t(<R>, w).



Applications of the Recursion Theorem

Applications of Recursion Theorem

- The Recursion Theorem can be used to show various negative results, e.g., undecidability results.
- Application 1: Acc_{TM} is undecidable
  - We already know this, but the Recursion Theorem provides a new proof.
  - Suppose for contradiction that D is a TM that decides Acc_{TM}.
  - Construct another machine R using self-reference (justified by the Recursion Theorem):
- R: On input w:
  - Obtain < R > (using Recursion Theorem)
  - Run D on input <R, w> (we can construct <R, w> from <R> and w)
  - Do the opposite of what D does:
    - If D accepts <R, w> then reject.
    - If D rejects <R, w> then accept.

Application 1: Acc_{TM} is undecidable

- Suppose for contradiction that D decides Acc_{TM}.
- R: On input w:
  - Obtain < R >
  - Run D on input <R, w>
  - Do the opposite of what D does:
    - If D accepts <R, w> then reject.
    - If D rejects <R, w> then accept.
- RT says that TM R exists, assuming decider D exists.
- Formally, to apply RT, use the 2-input machine T:
- T: On inputs <M> and w:
  - Run D on input <M, w>
  - Do the opposite of what D does:
    - If D accepts <M, w> then reject.
    - If D rejects <M, w> then accept.

Application 1: Acc_{TM} is undecidable

- Suppose for contradiction that D decides Acc_{TM}.
- R: On input w:
  - Obtain < R >
  - Run D on input <R, w>
  - Do the opposite of what D does:
    - If D accepts <R, w> then reject.
    - If D rejects <R, w> then accept.
- Now get a contradiction:
  - If R accepts w, then
    - D accepts <R, w> since D is a decider for Acc_{TM}, so
    - R rejects w by definition of R.
  - If R does not accept w, then
    - D rejects <R, w> since D is a decider for Acc_{TM}, so
    - R accepts w by definition of R.
- Contradiction. So D can't exist, so Acc_{TM} is undecidable.

Applications of Recursion Theorem

- Application 2: $Acc01_{TM}$ is undecidable
  - Similar to the previous example.
  - Suppose for contradiction that D is a TM that decides $Acc01_{TM}$.
  - Construct another machine R using the Recursion Theorem:
- R: On input w: (ignores its input)
  - Obtain < R > (using RT)
  - Run D on input <R>
  - Do the opposite of what D does:
    - If D accepts <R> then reject.
    - If D rejects <R> then accept.
- RT says that R exists, assuming decider D exists.

Application 2: $Acc01_{TM}$ is undecidable

- Suppose for contradiction that D decides $Acc01_{TM}$.
- R: On input w:
  - Obtain < R >
  - Run D on input <R>
  - Do the opposite of what D does:
    - If D accepts <R> then reject.
    - If D rejects <R> then accept.
- Now get a contradiction, based on what R does on input 01:
  - If R accepts 01, then
    - D accepts <R> since D is a decider for $Acc01_{TM}$, so
    - R rejects 01 (and everything else), by definition of R.
  - If R does not accept 01, then
    - D rejects <R> since D is a decider for $Acc01_{TM}$, so
    - R accepts 01 (and everything else), by definition of R.
- Contradiction. So D can't exist, so $Acc01_{TM}$ is undecidable.

Applications of Recursion Theorem

- Application 3: Using Recursion Theorem to prove Rice's Theorem
  - Rice's Theorem: Let P be a nontrivial property of Turing-recognizable languages. Let $M_P = \{ < M > \mid L(M) \in P \}$. Then M_P is undecidable.
  - Nontriviality: There is some M_1 with $L(M_1) \in P$, and some M_2 with $L(M_2) \notin P$.
  - Implies lots of things are undecidable.
  - We already proved this; now, a new proof using the Recursion Theorem.
  - Suppose for contradiction that D is a TM that decides M_P.
  - Construct machine R using the Recursion Theorem:…

Application 3: Using Recursion Theorem to prove Rice's Theorem

- Rice's Theorem: Let P be a nontrivial property of Turing-recognizable languages. Let $M_P = \{ < M > \mid L(M) \in P \}$. Then M_P is undecidable.
- Nontriviality: $L(M_1) \in P$, $L(M_2) \notin P$.
- D decides M_P.
- R: On input w:
  - Obtain < R >
  - Run D on input <R>
  - If D accepts <R> then run M_2 on input w and do the same thing.
  - If D rejects <R> then run M_1 on input w and do the same thing.
- M_1 and M_2 are as above, in the nontriviality definition.
- R exists, by the Recursion Theorem.
- Get contradiction by considering whether or not $L(R) \in P$:

Application 3: Using Recursion Theorem to prove Rice's Theorem

- Rice's Theorem: Let P be a nontrivial property of Turing-recognizable languages. Let $M_P = \{ < M > \mid L(M) \in P \}$. Then M_P is undecidable.
- $L(M_1) \in P$, $L(M_2) \notin P$.
- D decides M_P.
- R: On input w:
  - Obtain < R >
  - Run D on input <R>
  - If D accepts <R> then run M_2 on input w and do the same thing.
  - If D rejects <R> then run M_1 on input w and do the same thing.
- Get contradiction by considering whether or not $L(R) \in P$:
  - If $L(R) \in P$, then
    - D accepts <R>, since D decides M_P, so
    - $L(R) = L(M_2)$ by definition of R, so
    - $L(R) \notin P$.

Application 3: Using Recursion Theorem to prove Rice's Theorem

- Rice's Theorem: Let P be a nontrivial property of Turing-recognizable languages. Let $M_P = \{ < M > \mid L(M) \in P \}$. Then M_P is undecidable.
- $L(M_1) \in P$, $L(M_2) \notin P$.
- D decides M_P.
- R: On input w:
  - Obtain < R >
  - Run D on input <R>
  - If D accepts <R> then run M_2 on input w and do the same thing.
  - If D rejects <R> then run M_1 on input w and do the same thing.
- Get contradiction by considering whether or not $L(R) \in P$:
  - If $L(R) \notin P$, then
    - D rejects <R>, since D decides M_P, so
    - $L(R) = L(M_1)$ by definition of R, so
    - $L(R) \in P$.
- Contradiction!

Applications of Recursion Theorem

- Application 4: Showing non-Turing-recognizability
  - Define MIN_{TM} = { < M > | M is a "minimal" TM, that is, no TM with a shorter encoding recognizes the same language }.
  - Theorem: MIN_{TM} is not Turing-recognizable.
  - Note: This doesn't follow from Rice:
    - Requires non-T-recognizability, not just undecidability.
    - Besides, it's not a language property.
  - Proof:
    - Assume for contradiction that MIN_{TM} is Turing-recognizable.
    - Then it's enumerable, say by enumerator TM E.
    - Define TM R, using the Recursion Theorem:
    - R: On input w: …

Application 4: Non-Turing-recognizability

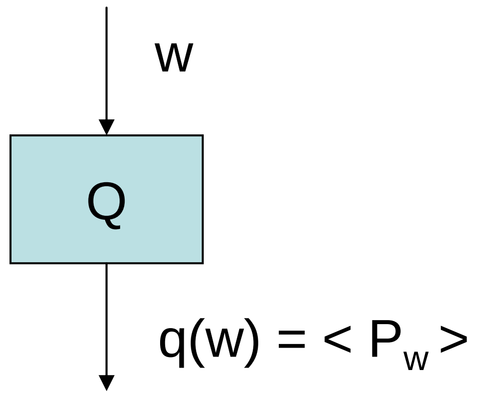
- MIN_{TM} = { < M > | M is a "minimal" TM }.
- Theorem: MIN_{TM} is not Turing-recognizable.
- Proof:
  - Assume that MIN_{TM} is Turing-recognizable.
  - Then it's enumerable, say by enumerator TM E.
  - R: On input w:
    - Obtain <R>.
    - Run E, producing list < M_1 >, < M_2 >, … of all minimal TMs, until you find some < M_i > with |< M_i >| strictly greater than |< R >|.
      - That is, until you find a TM with a rep bigger than yours.
    - Run M_i(w) and do the same thing.
  - Contradiction:
    - L(R) = L(M_i)
    - |< R >| less than |< M_i >|
    - Therefore, M_i is not minimal, and should not be in the list.

Proof of the Recursion Theorem: Special case

Proof of Recursion Theorem: Special Case

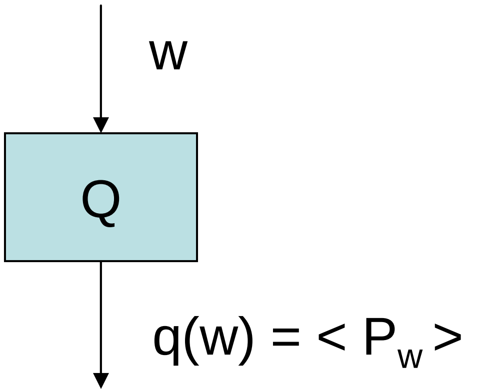
- Start with easier first step: Produce a TM corresponding to P_1:
- P_1:
  - Obtain < P_1 >
  - Output < P_1 >
- P_1 outputs its own description.

- Lemma: (Sipser Lemma 6.1): 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 just prints out w and halts.
- Proof: Straightforward construction. Can hard-wire w in the FSC of P_w.

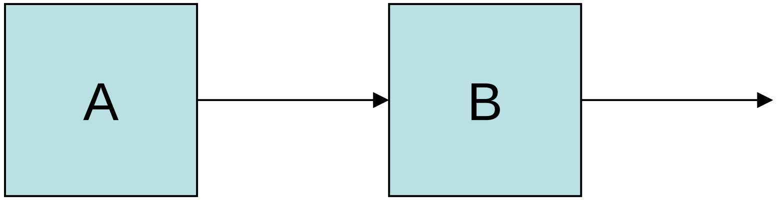


Proof of RT: Special Case

- Lemma: (Sipser Lemma 6.1): There is a computable function q: $\Sigma^* \to \Sigma^*$ such that, for any string w, q(w) is the description of a TM P_w that just prints out w and halts.

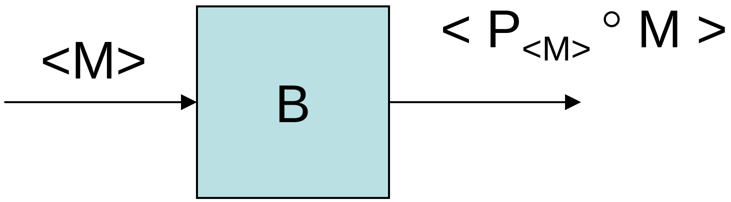


- Now, back to the machine that outputs its own description…
- Consists of 2 sub-machines, A and B.

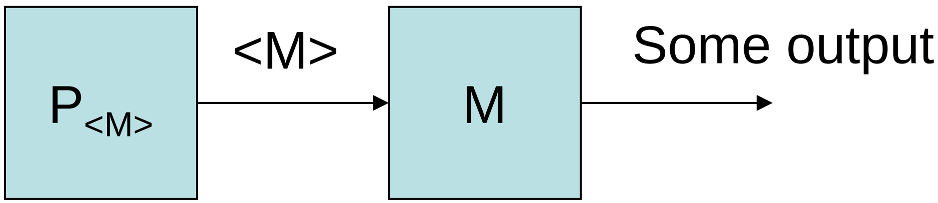


- Output of A feeds into B.
- Write as $A \circ B$.

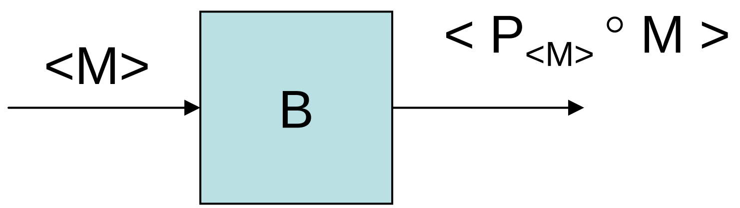
Construction of B



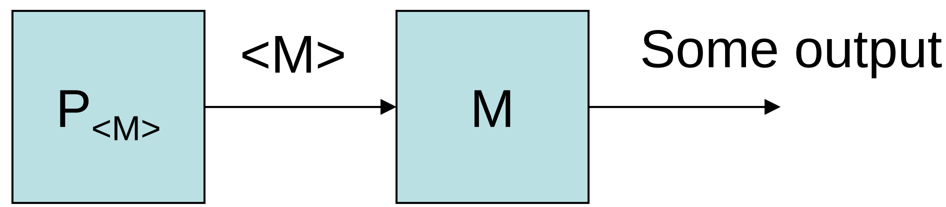
- B expects its input to be the representation <M> of a 1-input TM (a function-computing TM, not a language recognizer).
  - If not, we don't care what B does.
- B outputs the encoding of the combination of two machines, $P_{<M>}$ and M.
- The first machine is $P_{<M>}$, which simply outputs <M>.
- The second is the input machine M.
- $P_{<M>} \circ M$:



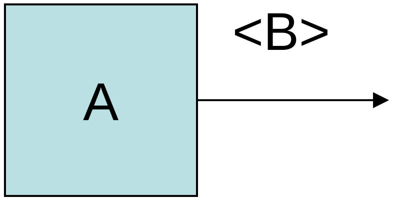
Construction of B



- How can B generate $< P_{<M>} \circ M >$?
  - B can generate a description of $P_{<M>}$, that is, $<P_{<M>}>$, by Lemma 6.1.
  - B can generate a description of M, that is, <M>, since it already has <M> as its input.
  - Once B has descriptions of $P_{<M>}$ and M, it can combine them into a single description of the combined machine $P_{<M>} \circ M$, that is, $< P_{<M>} \circ M >$.



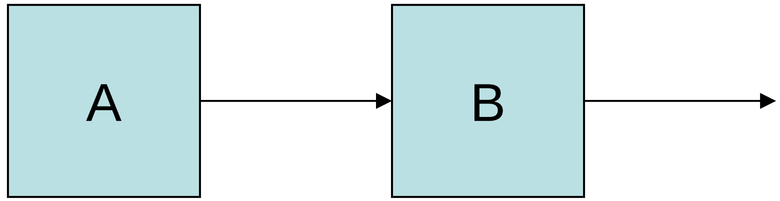
Construction of A



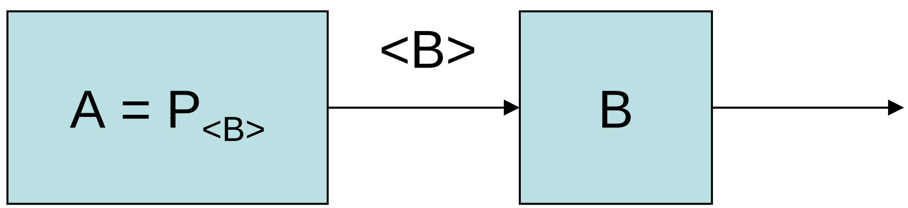
- A is $P_{<B>}$, the machine that just outputs <B>, where B is the complicated machine constructed above.
- A has no input, just outputs <B>.

Combining the Pieces

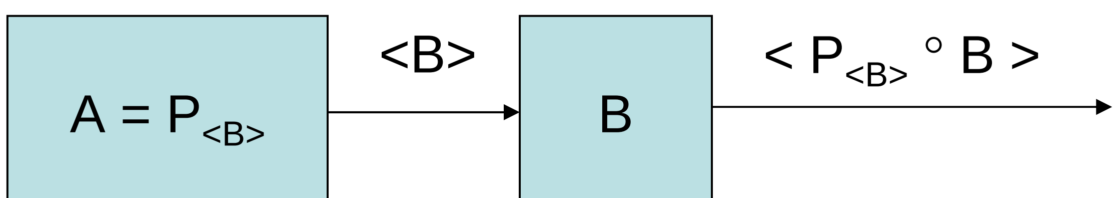
- A ∘ B:



- Claim A ∘ B outputs its own description, which is < A ∘ B >.
- Check this…
- A is $P_{<B>}$, so the output from A to B is <B>:

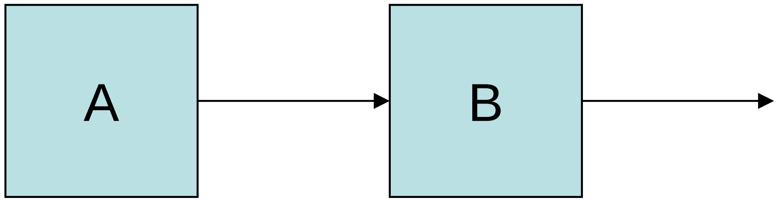


- Substituting B for M in B's output:

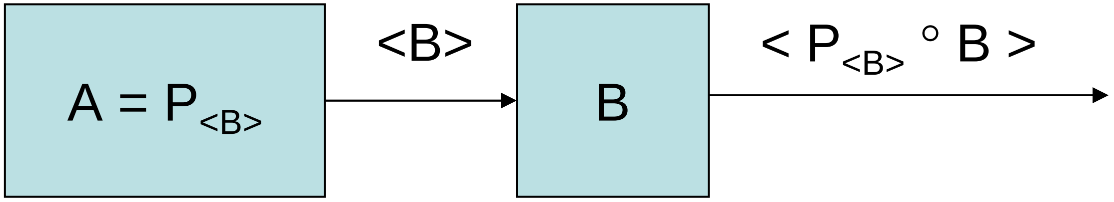


Combining the Pieces

- $A \circ B$:



- Claim $A \circ B$ outputs its own description, which is $< A \circ B >$.



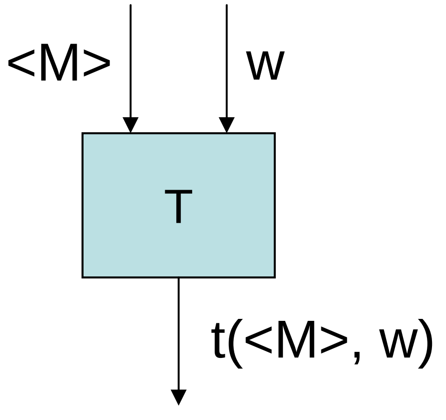
- The output of $A \circ B$ is, therefore, $< P_{<B>} \circ B > = < A \circ B >$.
- As needed!

- $A \circ B$ outputs its own description, $< A \circ B >$.

Proof of the Recursion Theorem: General case

Proof of the RT: General case

- So, we have a machine that outputs its own description.
- A curiosity---this is not the general RT.
- RT says not just that:
  - There is a TM that outputs its own description.
- But that:
  - There are TMs that can use their own descriptions, in “arbitrary ways”.
- The “arbitrary ways” are captured by the machine T in the RT statement.

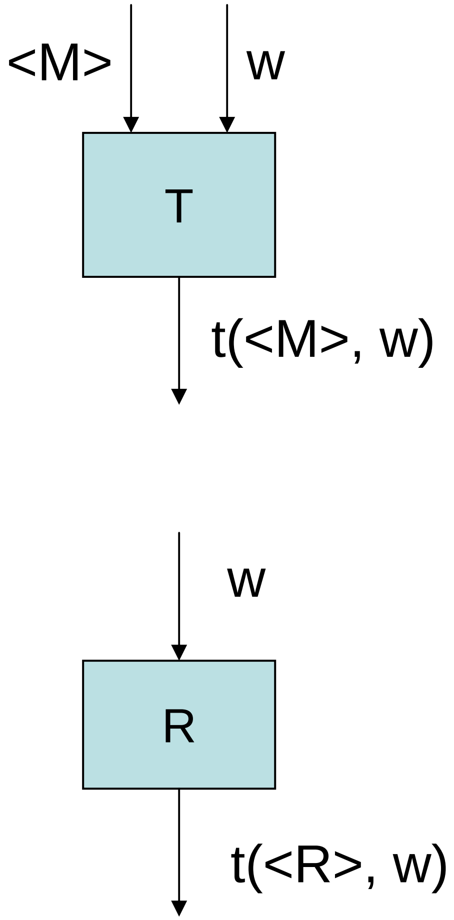


The Recursion Theorem

- Recursion Theorem:

  Let T be a TM that computes a (possibly partial) 2-argument function t: $\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$.

  Then there is another TM R that computes the function r: $\Sigma^* \rightarrow \Sigma^*$, where for any w, r(w) = t(<R>, w).



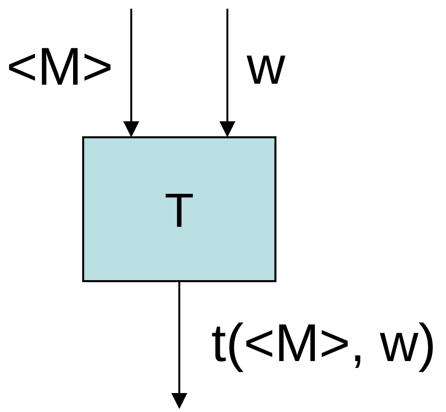
The Recursion Theorem

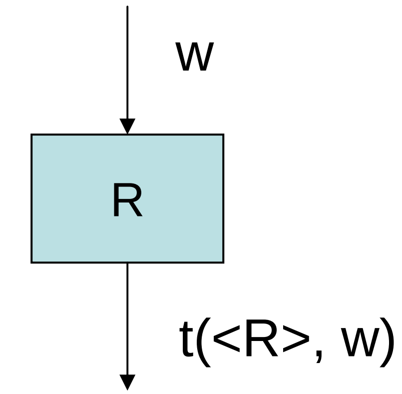
- Recursion Theorem:

  Let T be a TM that computes a (possibly partial) 2-argument function t: $\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$.

  Then there is another TM R that computes the function r: $\Sigma^* \rightarrow \Sigma^*$, where for any w, r(w) = t(<R>, w).

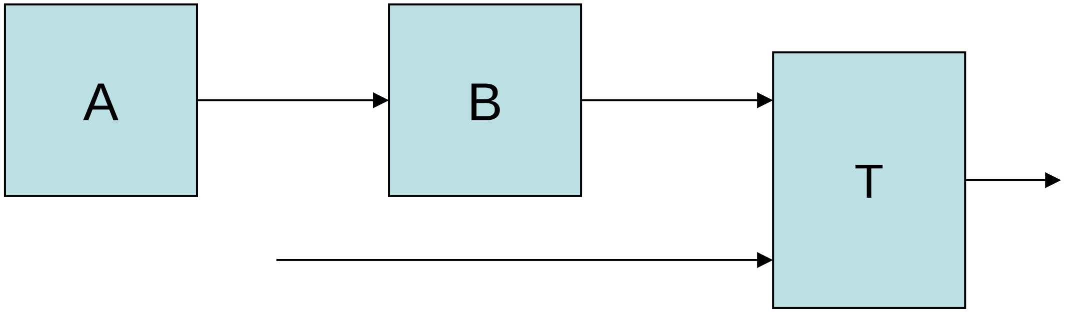
- Construct R from:
  - The given T, and
  - Variants of A and B from the special-case proof.





Proof of RT: General Case

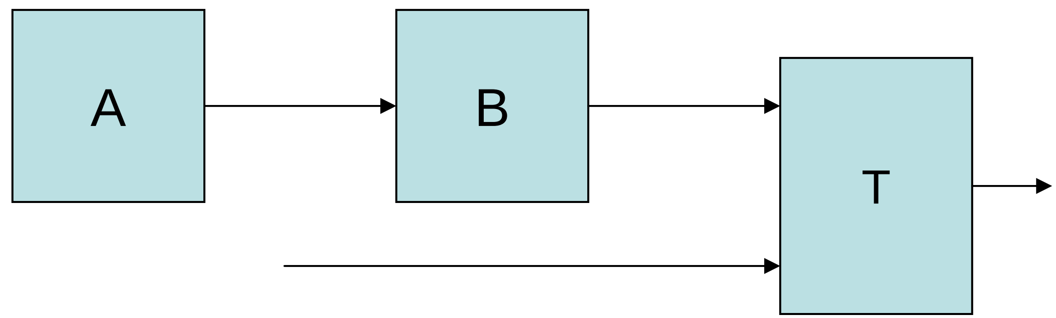
- R looks like:



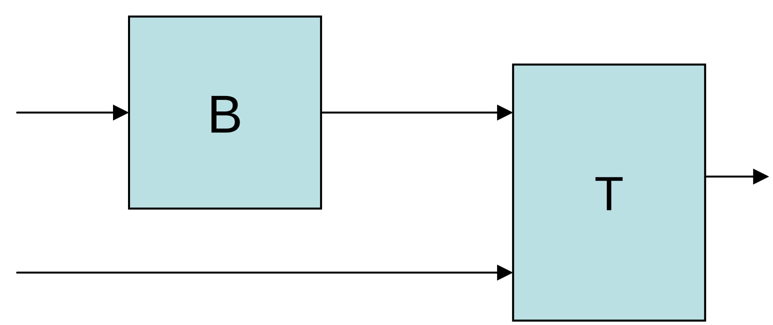
- Write this as $(A \circ B) \circ_1 T$
  - The $\circ_1$ means that the output from $(A \circ B)$ connects to the first (top) input line of T.

Proof of RT: General Case

- $R = (A \circ B) \circ^1 T$

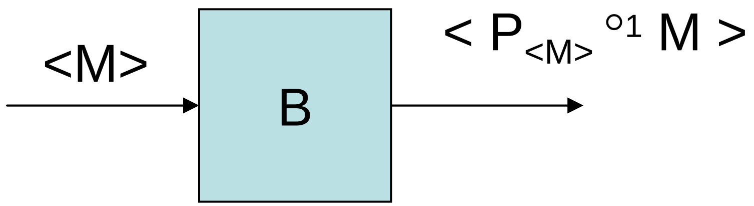


- New A: $P_{<B \circ^1 T>}$, where $B \circ^1 T$ means:

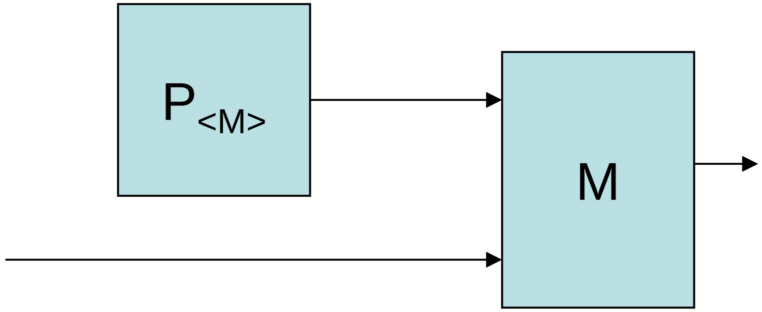


Proof of RT: General Case

- New B:

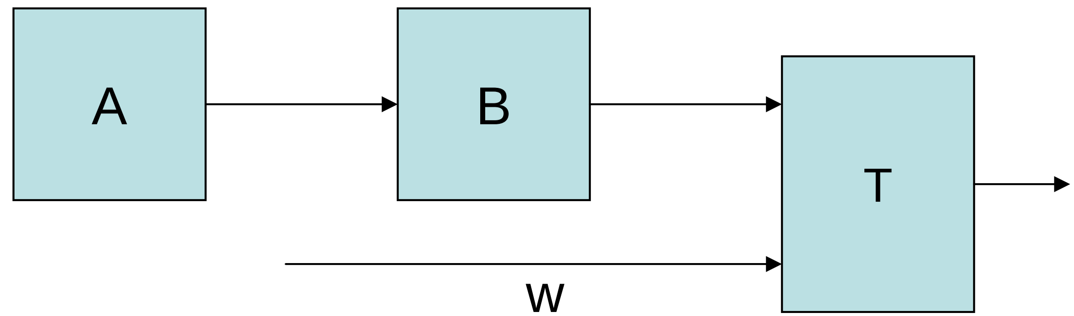


- Like B in the special case, but now M is a 2-input TM.
- $P_{<M>} \circ_1 M$: 1-input TM, which uses output of $P_{<M>}$ as first input of M.

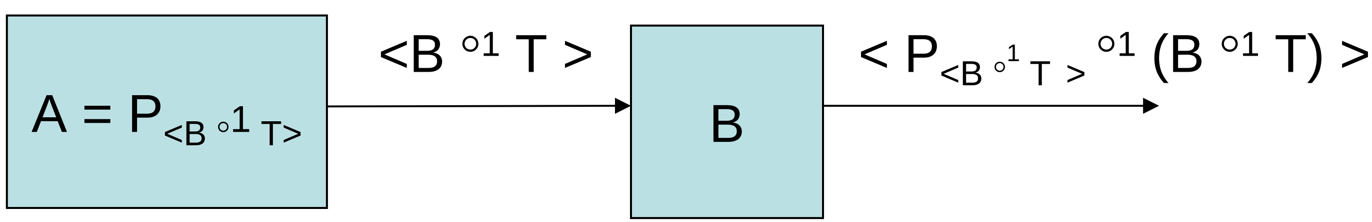


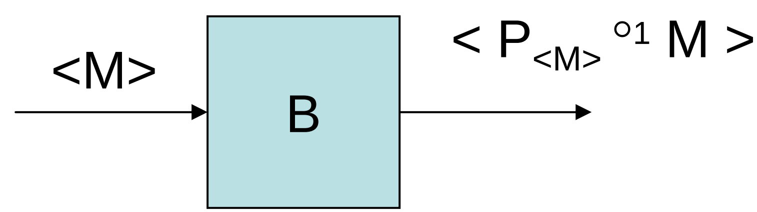
Combining the Pieces

- $R = (A \circ B) \,^{\circ 1}\, T$



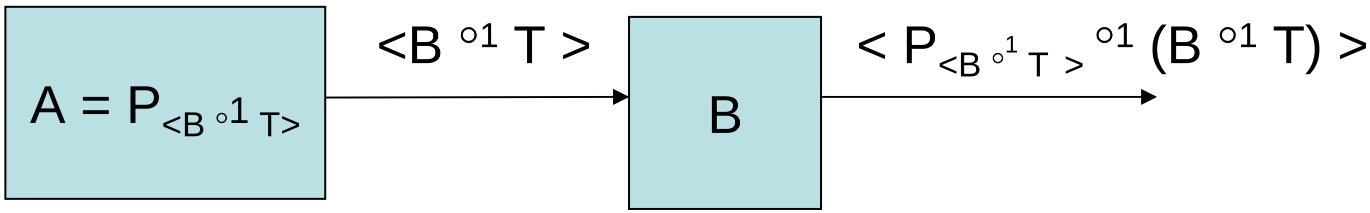
- Claim R outputs $t(\langle R \rangle, w)$:
- A is $P_{\langle B \,^{\circ 1}\, T \rangle}$, so the output from A to B is $\langle B \,^{\circ 1}\, T \rangle$:



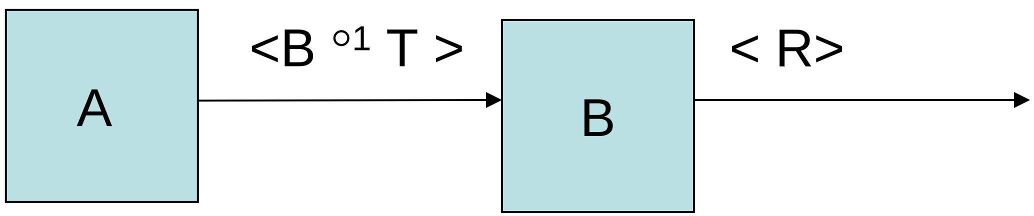


- Now recall definition of B:
- Plug in $B \,^{\circ 1}\, T$ for M in B's input, and obtain output for B.

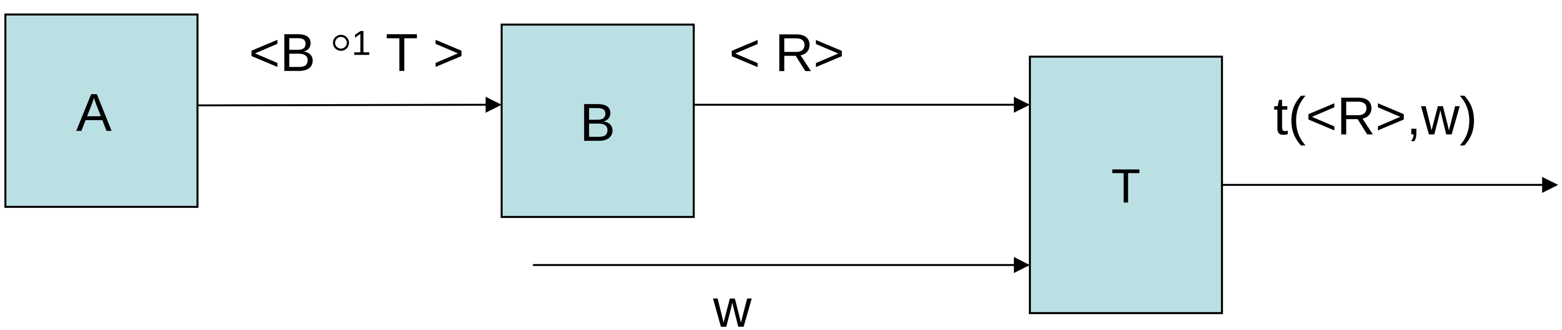
Combining the Pieces



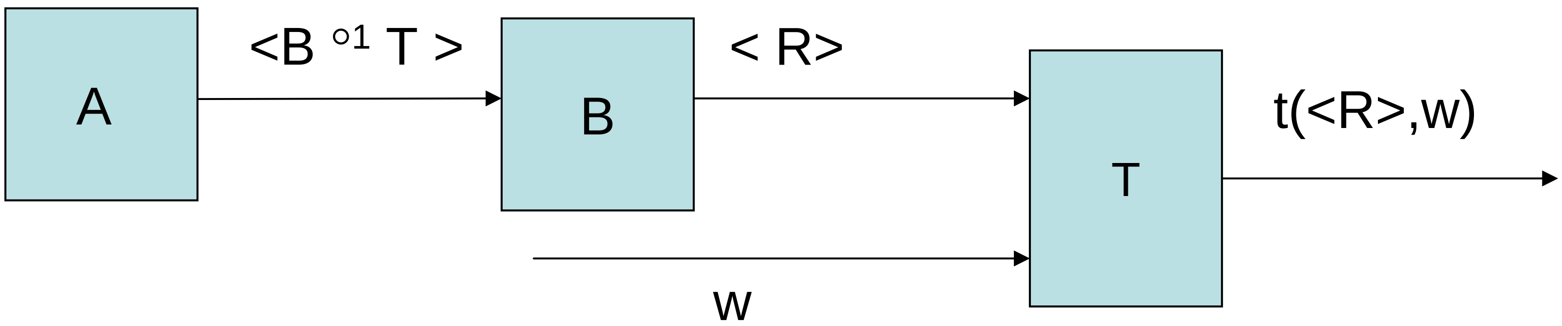
- B's output = $< A \circ 1 \ (B \circ 1 \ T) > \ = \ < R >$:



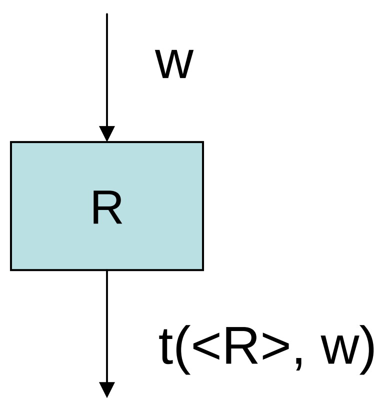
- Now combine with T, plugging in R for M in T's input:



Combining the Pieces



- Thus, $R = (A \circ B) \circ_1 T$, on input w, produces t(<R>,w), as needed for the Recursion Theorem.



Next time…

- More on computabilty theory
- Reading:
  - "Computing Machinery and Intelligence" by Alan Turing:

    http://www.loebner.net/Prizef/TuringArticle.html

6.045J / 18.400J Automata, Computability, and Complexity
Spring 2011