Last time: Church-Turing Thesis in other words:

"Turing machines precisely capture the intuitive notion of what we think of as computation"

Input encoding:

many problems have multiple input encodings

eg. Graphs $G = (V, E)$

- adjacency matrix
- adjacency lists (eg. singly, doubly linked)
- edge lists

a TM can convert any one of these encodings to the other so we don't care which one.

This is a string over some fixed alphabet

Use $\langle G \rangle$ to denote any reasonable encoding

angle brackets in latex $\langle G\rangle

$$\{\langle G \rangle \mid G \text{ is a graph and} \dots\}$$

Describe TM operating on such an input like any high level graph algorithm.

Encoding multiple objects in a single input string use comma to separate

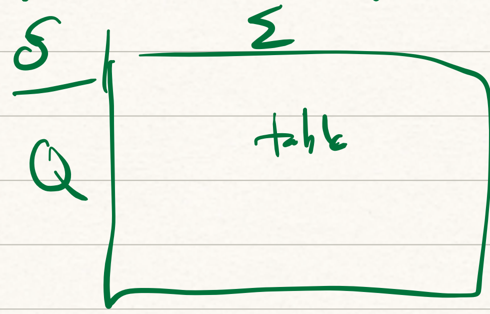
eg $\langle x, y \rangle$ "encoding of x followed by encoding of y in a way you can decode them"

Another example: DFAs

Recall a DFA M has

$Q, \Sigma, \delta, q_0, F$ — $q_0 \in Q$, $F \subseteq Q$

- Q: set of states
- Σ: set of symbols
- δ: transition function $\delta: Q \times \Sigma \to \Sigma$
- q_0: start state
- F: set of accepting/final state



- encoding is similar to that of graphs

$L(M) = \{ w \in \Sigma^* : M \text{ accepts } w \}$

Use $\langle M \rangle$ to denote a natural encoding of a DFA ← it's just a labelled graph

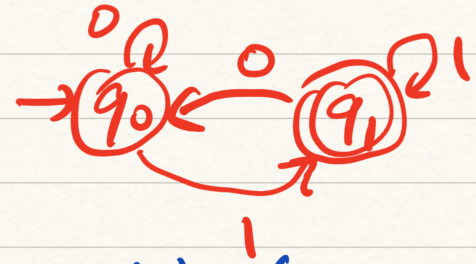
Note: The DFA's alphabet Σ may be much larger than the alphabet used for $\langle M \rangle$:

- we can assume that the alphabet used for $\langle M \rangle$ is $\{0, 1, \#\}$ for convenience
- it will take several symbols over $\{0, 1, \#\}$ to encode a single state name or symbol of M

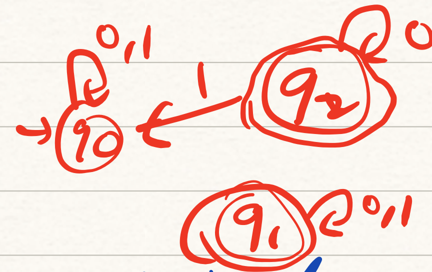
Questions we might want to know about a DFAs...

- Does DFA M accept anything? — $E_{TM} = \{ \langle M \rangle : M \text{ is a DFA s.t. } L(M) = \emptyset \}$
- Are two DFAs M_1, M_2 equivalent? — $EQ_{TM} = \{ \langle M_1, M_2 \rangle : M_1, M_2 \text{ are DFAs with } L(M_1) = L(M_2) \}$
- Does DFA M accept a specific input w? — $A_{TM} = \{ \langle M, w \rangle : M \text{ is a DFA that accepts string } w \}$

e.g.



$L(M) \neq \emptyset$



$L(M) = \emptyset$

Thm E_{DFA} is decidable

Proof For a DFA, $L(M)$ is empty iff no path from the start state q_0 to any of the states in F

Algorithm: Do a graph search (DFS, BFS, ...) in diagram of M starting at q_0
- if state in F reached, reject
- if no state in F reached accept

∎

$A_{DFA} = \{\langle M, w\rangle : M$ is a DFA that accepts string $w\}$

Thm A_{DFA} is decidable

Proof: TM

Simulate M on input w step by step for $|w|$ steps until DFA reaches end of w.

- keep pointer on next char to be read (on tape)
- record current state of M (on tape)

need this since M might have more states than the TM

Accept if state in F reached at end of w

∎

Note: Simulate M is different from run M

"use M as a subroutine"

Run M would mean incorporating M into the defn of the TM like we did with your address incrementing solution when we created an NTM equivalent to a TM.

We can't do this because M may have more states and symbols than the decider which has to work for all possible M.

Instead we use the tapes to store the state and encode the symbols.

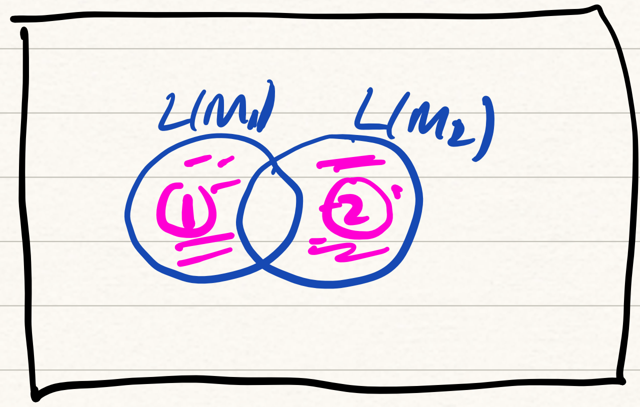
Thm EQ_{DFA} is decidable

Proof We give two algorithms based on
1. closure properties of DFAs
2. minimization alg for DFAs

$\boxed{\text{Closure}}$

Recall from 311

Given DFAs M, M' we can build new DFA for
- complement $\overline{L(M)}$
  change F to $Q-F$
- $L(M) \cup L(M')$
  $L(M) \cap L(M')$
  } cross-product construction



$L(M_1) = L(M_2)$ iff ① ∪ ② is empty.

① $= L(M_1) \cap L(\overline{M_2})$
② $= L(M_2) \cap L(\overline{M_1})$

Algorithm I: Build DFA ⟨M⟩ s.t.

$$L(M) = \left(L(M_1) \cap L(\overline{M_2})\right) \cup \left(L(M_2) \cap L(\overline{M_1})\right)$$

- Run decider for E_{DFA} on input ⟨M⟩ and output its answer

Minimization: Recall DFA minimization alg from 311.
We didn't prove it but it turns out that if $L(M_1) = L(M_2)$
then the minimized versions of M_1 and M_2 will be the same up to renaming of states (which is easy to check since edge labels need to match)

Algorithm 2: • Run minimization alg on M_1 and on M_2

• Check that minimized forms are the same (up to state renaming)

Note • This second alg is efficient in practice (much more than the first)

• In fact this second algorithm was what was used to grade your CSE 311 finite state machine homework!

$\boxtimes$

We can define the analogous languages for answering the same questions for NFAs

e.g. $E_{NFA} = \{\langle M \rangle : M \text{ is an NFA and } L(M) = \emptyset\}$

A_{NFA}, EQ_{NFA}.

Thm E_{NFA} is decidable

Proof Algorithm 1: • Convert $\langle M \rangle$ to $\langle M' \rangle$ where M' is a DFA with $L(M') = L(M)$ using subset construction
• Run decider for E_{DFA} on input $\langle M' \rangle$

note size of M' is exponential in size of M but that's ok.

Algorithm 2: (Same as algorithm for E_{DFA} on diagram of M
check if state of F reachable from q_0 ∎

$A_{NFA} = \{ \langle M, w \rangle : \text{NFA } M \text{ accepts } w \}$

Thm A_{NFA} is decidable

Proof

Algorithm: • Convert input $\langle M, w \rangle$ for NFA M to $\langle M', w \rangle$ for equivalent DFA M'
• Run decider for A_{DFA} on $\langle M', w \rangle$ ∎

Can do something similar for EQ_{NFA}.

Can also decide similar properties for regular expressions
e.g. A_{REG}, EQ_{REG}

$A_{REG} = \{\langle R, w\rangle :$ R is a regular expression that generates $w\}$

Thm A_{REG} is decidable

Proof On input $\langle R, w\rangle$ convert R to an equivalent NFA M and run decider for A_{NFA} on input $\langle M, w\rangle$

$\boxtimes$

Recall.

Context-Free Grammar

Given by

V - finite set of variables

Σ - alphabet (terminals)

R set of rules of form

$$A \to w \qquad A \in V$$
$$w \in (V \cup \Sigma)^*$$

$S \in V$ start symbol

$A \Rightarrow_G^* w$ repeatedly apply rules from A to get w

$L(G) = \{ w \in \Sigma^* \mid A \overset{*}{\Rightarrow}_G w \}$

CSE 311 Examples: • Set of binary palindromes

$S \to \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

• Strings of balanced parentheses

$S \to \varepsilon \mid (S) \mid SS$

• strings with equal #'s of 0's and 1's.

$S \to \varepsilon \mid 0S1 \mid 1S0 \mid SS$

$E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \}$

Thm E_{CFG} is decidable

Examples

$\underbrace{S \to 0S0 \mid 1S1}_{\text{empty}}$

$S \to 0SA \mid 1S1$
$\underbrace{A \to 0}_{\text{empty}}$

$S \to 0BA \mid 1S1$
$\underbrace{A \to 0, \quad B \to 1}_{\text{not empty.}}$

Proof Idea: keep track of all symbols that can produce strings of terminals.

Call a variable $A \in V$ "productive" iff $A \Rightarrow^* w$ for some $w \in \Sigma^*$

Let P be the set of productive variables

Can compute P as follows:

- Put all variables A with rule $A \to w$ for $w \in \Sigma^*$, into P
- Repeatedly:
  Add $A \in V$ to P if there is a rule $A \to w$ with $w \in (\Sigma \cup P)^*$

This algorithm to compute P will stop since G is finite.

Algorithm for E_{CFG}:
On input $\langle G \rangle$
- Compute P
- if $S \in P$ reject if $S \notin P$ accept.

$\boxtimes$

<u>Fact:</u> $A_{CFG} = \{\langle G, w \rangle : G \Rightarrow^* w\}$ is decidable

Not obvious since rules might produce intermediate strings much larger that the final string w.
We will give a polytime alg later $\boxtimes$

We will see that EQ_{CFG} is not decidable
but that is getting ahead of things!

Now to TMs

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

<u>Thm</u> A_{TM} is Turing-recognizable

This is a weaker statement than decidable
if no, can either reject
<u>or</u> run forever

<u>Proof</u> (Turing's idea)

Algorithm: Universal Turing Machine U

Turing machine simulator

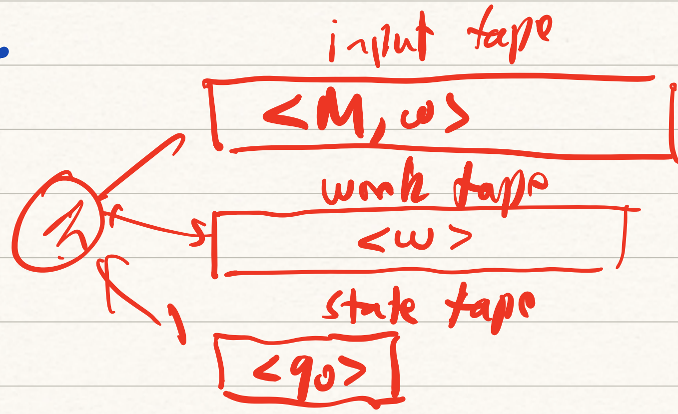
U: On input $\langle M, w\rangle$

TM description/diagram

perform a step-by-step simulation of M on input w.

If: M accepts then accept
M rejects then reject

Details of U:



- start by putting $\langle w \rangle$ on work tape and $\langle q_0 \rangle$ on state tape
- maintain encoding of current tape (and head position) on work tape and current state on state tape.

  use δ table on input to figure out how to update work tape and state tape
- accept iff M does
- reject iff M does.

$\boxtimes$

Thm A_{TM} is <u>not</u> decidable

<u>Proof</u> idea diagonalization.

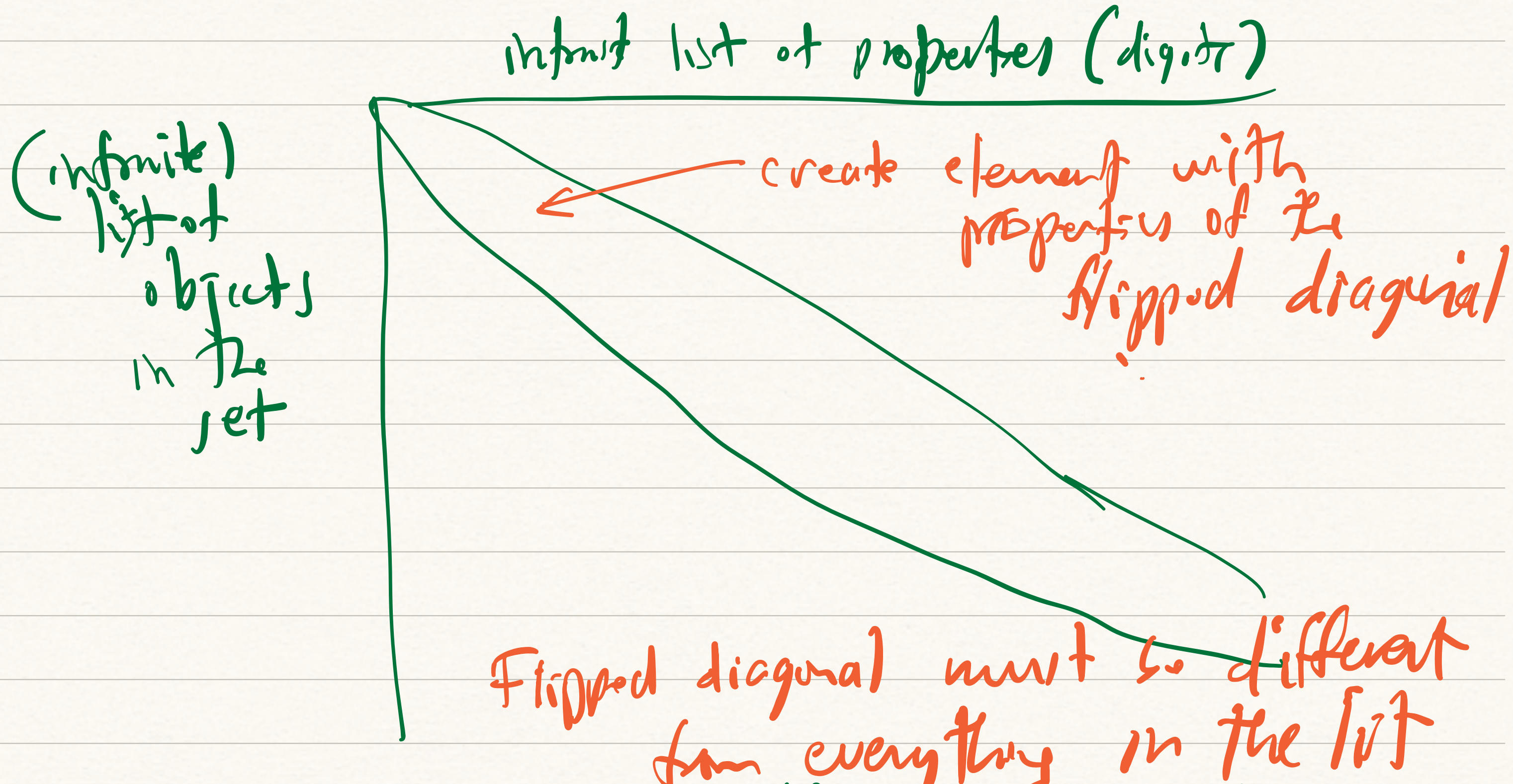
Before the actual proof, some intuition...

Recall the CSE311 proofs that certain sets are not countable e.g. reals

S is <u>countable</u> iff we can list S as

$$S = \{s_1, s_2, s_3, \ldots\}$$

The idea was to assume for contradiction that some set e.g. $\mathbb{R}^{(0,1)}$ is countable and build an infinite table



so list is not complete

In these proofs the contradiction will be to the assumption that the list had everything.

In Turing's proof we will think of a row for each TM M.

The set of TMs is <u>countable</u> since

$\{\langle M\rangle : M \text{ is a TM}\} \subseteq \Sigma^*$ for some Σ

We can simply list the TMs in order of $\langle M\rangle$

The contradiction instead will be to being able to fill out the table & flip.

We can have a table entry for every input pair $\langle M, w\rangle$ but we only need to worry about some w

(*) Suppose that some TM H decides A_{TM}.

Focus on strings w that are codes of TMs

Can list all TMs since set of TMs is countable

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	...	
D	0	0	1	1	1	
M_1	1	0	1	1	0	?
M_2	0	1	0	0	1	0
M_3	1	1	0	1	0	0
M_4	0	1	0	0	0	1 ...
M_5	1 ...	0	1	1	0	...
⋮						

$$(i,j) \text{ entry} = \begin{cases} 1 & \text{iff } M_i \text{ accepts } \langle M_j\rangle \text{ iff } H \text{ accepts } \langle M_i, \langle M_j\rangle\rangle \\ 0 & \text{iff } M_i \text{ does not accept } \langle M_j\rangle \text{ iff } H \text{ rejects } \langle M_i, \langle M_j\rangle\rangle \end{cases}$$

Given the TM H, this motivates defining a TM D for the flipped diagonal language as follows:

D: On input $\langle M \rangle$:

Let $w = \langle M \rangle$

Run H on input $\langle M, w \rangle$: $\langle M, \langle M \rangle \rangle$

if H accepts then reject

if H rejects then accept

For any i:

Since D behaves differently from M_i on input $\langle M_i \rangle$, $D \neq M_i$

However, by construction the listing of TMs $M_1, M_2, \ldots$ was a complete list which is a contradiction to our assumption

⊛

$\therefore$ A_{TM} is not decidable.

Note that we don't need the table at all to do the proof, just the intuition

So we just write it directly:

Proof: (by Contradiction)

Suppose that A_{TM} is decidable

Then there is some TM H that decides A_{TM}.

Define a new TM D as follows:

On input $\langle M \rangle$ that is the code of a TM

Create the string $\langle M, \langle M \rangle \rangle$

Run H on $\langle M, \langle M \rangle \rangle$

if H accepts then reject

if H rejects then accept

we don't really care what D does on other strings but we could have it always reject them.

Consider running D on input $\langle D \rangle$ "i.e. on itself"

Now D accepts $\langle D \rangle$

$\Leftrightarrow$ H accepts $\langle D, \langle D \rangle \rangle$ by defn of H

$\Leftrightarrow$ D does not accept $\langle D \rangle$ by defn of D

contradiction

We can have some immediate consequences:

Recall that:

Thm If L is decidable then $\overline{L}$ is decidable

swap q_{acc} and q_{rej}

Thm If L is T-rec and $\overline{L}$ is T-rec

then L is decidable

two tape run in parallel.

Cor $\overline{A_{TM}}$ is not Turing recognizable

Proof A_{TM} is T-rec.

If $\overline{A_{TM}}$ also was T-rec then

A_{TM} would be decidable (which it is not). ∎

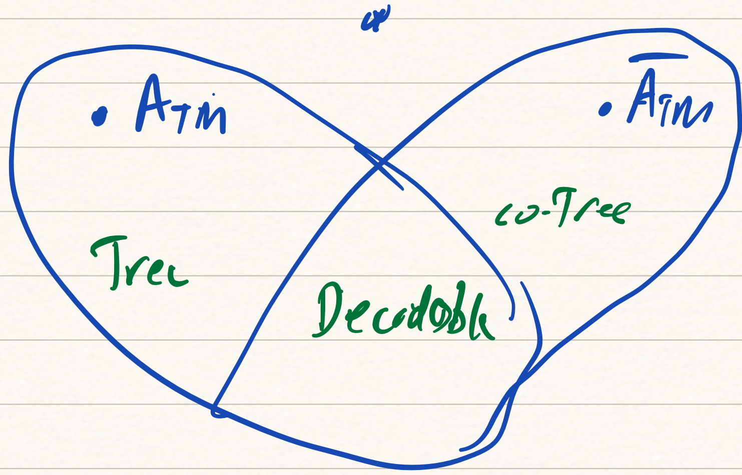
Def^n A is <u>co-Turing recognizable</u>

iff $\overline{A}$ is Turing-recognizable

We have the following picture of the space of languages over Σ.

A_{TM} is co-T-rec

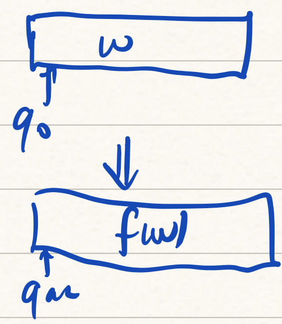
← we will also see languages out here



We can now prove lots of other problems undecidable.

To do this we introduce a general methodology (see Sipser sect. 5.3) for which we need some definitions

Def^n A TM M computes a function $f : \Sigma^* \to \Sigma^*$ iff for all $w \in \Sigma^*$ given to M as input M halts on 1st cell of the tape with $f(w)$ on tape followed by all blanks.



e.g. your HW1 problem asked you to produce a TM that computes the lexographically next string function.

Def^n We say that f is <u>computable</u> iff there is some TM that computes it

<u>Mapping Reductions</u>

<u>Def^n</u> Given $A, B \subseteq \Sigma^*$ we say that A is <u>mapping reducible</u> to B iff $\exists$ computable function $f: \Sigma^* \to \Sigma^*$ s.t. $\forall w \in \Sigma^*$

mapping reduction

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

<u>Notation</u> : $A \leq_m B$

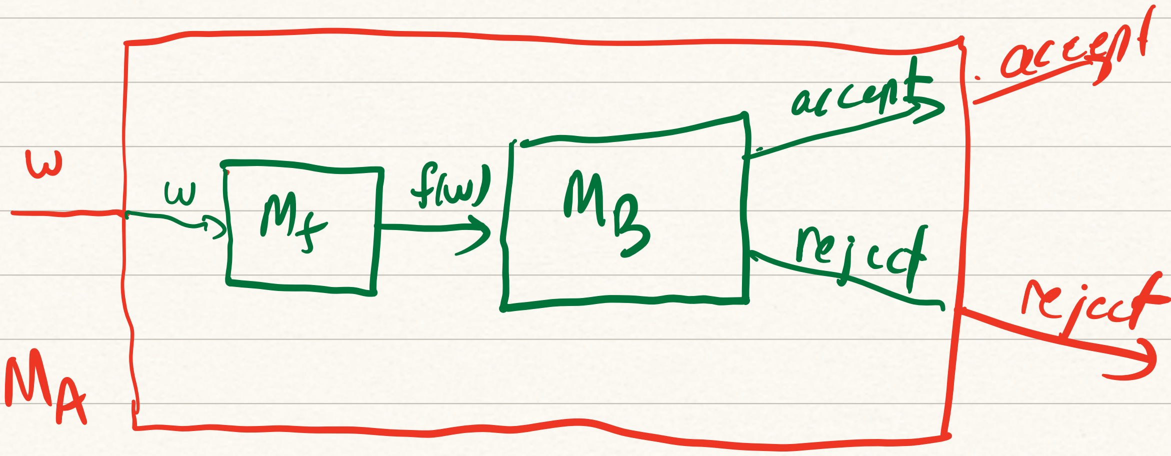
Idea: Roughly : A is (roughly) as easy as B
B is (roughly) as hard as A
Let's make this precise :

<u>Thm</u> (a) If $A \leq_m B$ and B is decidable then A is decidable.
(b) If $A \leq_m B$ and B is T-rec then A is T-rec

<u>Proof</u> (a) Since B is decidable there is a decider TM M_B for B

Since $A \le_m B$ there is a function f computable by some TM M_f s.t. $\forall w \in \Sigma^*$ $w \in A \iff f(w) \in B$.

We build a decider M_A for A as follows



correctness follows directly:
$w \in A \iff f(w) \in B \iff M_B$ accepts $f(w) \iff M_A$ accepts w

Since M_A is a decider ; M_B is a decider

(b) Use a recognizer TM M_B instead of a decider
The same construction works correctly for the same reason. ∎

we will use this a lot

Cor If $A \le_m B$ and A is <u>not</u> decidable then B is not decidable

If $A \le_m B$ and A is <u>not</u> T-rec the B is not T-rec.