Lecture 6

Frank

February 21, 2022

1 Rice's Theorem

1.1 Introduction

Property P of (recognizable) languages is a set of (recognizable) languages with that property P.

- P is trivial if $P = \emptyset$ or $P =$ all recognizable languages
- Beware of the difference between $P = \emptyset$ and $P = \{\emptyset\}$

  $P = \emptyset$ is trivial while $P = \{\emptyset\}$ is nontrivial
- P is a property, and $\bar{P}$ is also a property, namely the property of not having property P
- Note we may have $L, \bar{L} \in P$, namely the property of being either in the language or its complement.

1.2 Theorem

- **Thm 6.1 (Rice's Theorem)**: Let P be any nontrivial property of recognizable languages and let $T_P = \{\langle M \rangle : L(M) \in P\}$.

  $\implies T_P$ is undecidable

  Proof:

  P is any non-trivial property, so $\emptyset \in P$ or $\emptyset \notin P$

  Want to show $U \leq_m T_P$

  want computable $f : \langle M, x \rangle \to \langle M' \rangle$ s.t.

  a) M accepts $x \implies L(M') \in P$

  b) M doesn't accept $x \implies L(M') \notin P$

  Case 1: $\emptyset \notin P$

  Fix $L \in P$ [exists because P is non-trivial]

  and let M_L be a TM that recognizes L.

Then f is

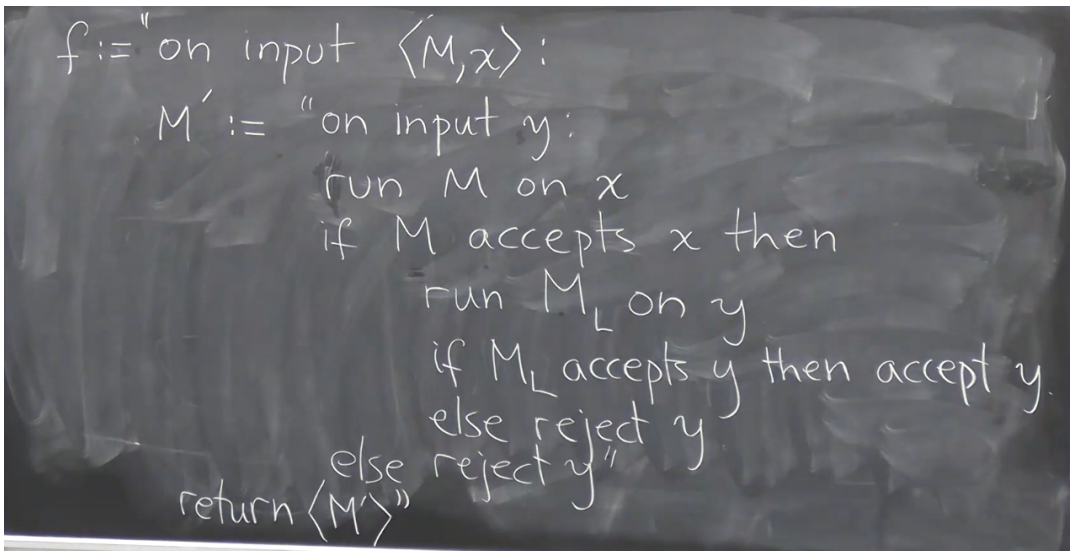


Figure 1: l71

The idea is that if M accepts x, then we want to accept, loop on, or reject the exact same strings as M_L. Then obviously M' accepts the same language as M_L, which is $L \in P$. So $L(M')$ has property P.

On the other hand, if M does not accept x, then we want to accept nothing $[L(M') = \emptyset \notin P]$, then M' does not have property P.

Here is a graphical representation of M'

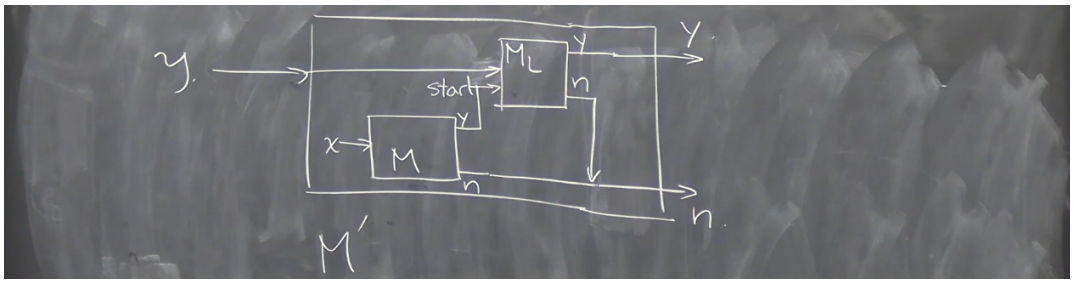


Figure 2: l72

Case 2: $\emptyset \in P$

$\implies \emptyset \notin \bar{P}$ [case 1]

$\implies T_{\bar{P}}$ is undecidable

$\implies \bar{T}_{\bar{P}}$ is undecidable [thm 3.x]

Which is the set of codes of TMs whose languages don't have property $\bar{P}$, which is the set of codes of TMs whose languages have property P

Completing our proof

1.3 Limitations of Rice's Theorem

1. It's about undecidability, not unrecognizability
2. It's about (codes of) TMs that do something, so it only applies to the form $\{\langle M\rangle : L(M)\ldots\}$
3. It's about the languages that the TMs recognize. For example, we could have some "syntactic" properties about the TMs, in which case this theorem will be unapplicable.

   Not about:

   - "syntactic" properties
   - behaviour of the TM

2 More Reductions

2.1 Example 1

Let $EQUIV = \{\langle M_1, M_2\rangle : L(M_1) = L(M_2)\}$

- **Thm 6.2**: EQUIV is unrecognizable

  Want

  $\bar{U} \leq_m EQUIV$
  or $U \leq_m EQ\bar{U}IV$

  But we can simplify this problem by instead reducing another language to EQUIV

  Let $E = \{\langle M\rangle : L(M) = \emptyset\}$

  Show that $E \leq_m EQUIV$ easily

  Just do $\langle M\rangle \to \langle M, M_\emptyset\rangle$, and run $\langle M, M_\emptyset\rangle$ on the decider of $EQUIV$. Ex. we are simply testing if the language accepted by M is equivalent to the empty language.

2.2 Example 2

Let $SUBSET = \{\langle M, M'\rangle : L(M) \subseteq L(M')\}$

We can see that $E \leq_m SUBSET$ easily ($\emptyset$ can only be subset of $\emptyset$)

We can also do $EQUIV \leq_T SUBSET$ by using SUBSET twice (once subset of, and once reverse to guarantee equality)

3 Recursion Theorem

3.1 TMs as function computers

- Input: as before
- Steps: as before
- No longer have accept/reject states, instead we have halt state.
- If TM enters halt state, it stops and what's on the tape is the output
- If TM never halts on some input x, value of (partial) function it computes is undefined at that input x

3.2 Informal Definition of RT

Does endowing TMs with "self awareness" make them more powerful?

> self awareness is knowing and using its own description (ex. $\langle M \rangle$)
>
> For example, functions can call itself recursively, but we've never seen TMs call itself recursively. If it has the ability to do so, would that change anything?
>
> But by Church's Thesis, TMs should be able to do anything a computer can do. Thus, TMs should be able to do recursion.
>
> Any recursive program can be iterated instead.

Define program $P1$ (Knowing)

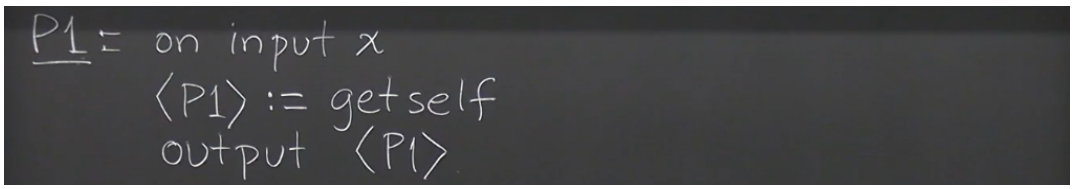


Figure 3: l6

Thus $P1$ is a program that prints itself.

Now define P2 (Knowing + Using)

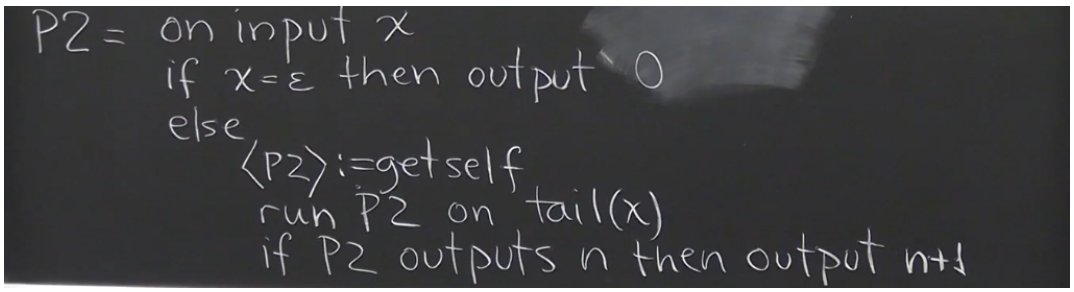


Figure 4: l73

Where tail(x) takes input string x and returns everything except the first element.

Then this program outputs the length of x. Note this function is only defined at $x = \epsilon$

Thm 6.3: Let T be a TM that computes function $t : \Sigma^\star \times \Sigma^\star \to \Sigma^\star$, then $\exists$ TM R that computes function $r : \Sigma^\star \to \Sigma^\star$ s.t. $r(x) = t(\langle R \rangle, x)$

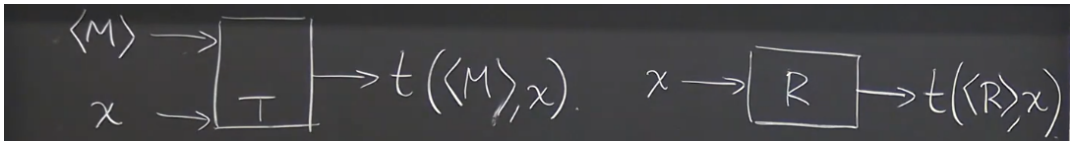


Figure 5: l75

This means that if we can compute something "knowing" the code of R, then we can compute the exact same thing without being given the code of R. Which means R can figure out its own code

This theorem justifies the statement "getself" inside our TM's algorithm

For P2:

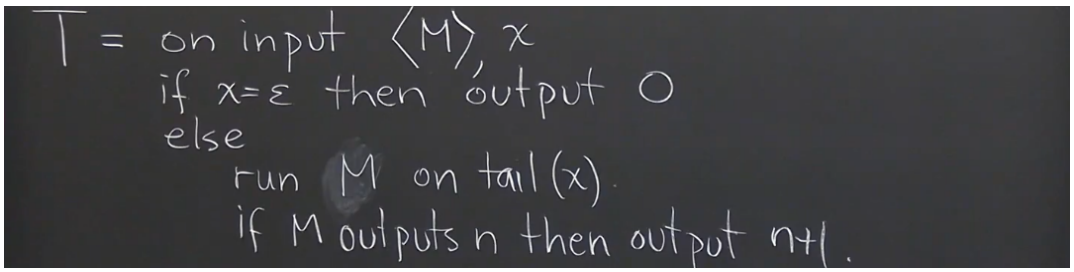


Figure 6: l76

Note that what T does is dependent on what M we feed to it. In other words, if we give it some trivial TM M that returns 7 on any x, then given $x = \epsilon$, we output 0, but if we give it a different x, we will return 8. If given a different M, we will do something else.

Recursion theorem says that there is an R that computes $t(\langle R \rangle, x)$. Or that there is a TM that can compute what T computes without being given explicitly, the encoding of a TM.

3.3 Revisiting Old Proofs Using RT

U is undecidable (via RT):

Suppose for contra, that U is decidable

Let U-DEC be decider for U.

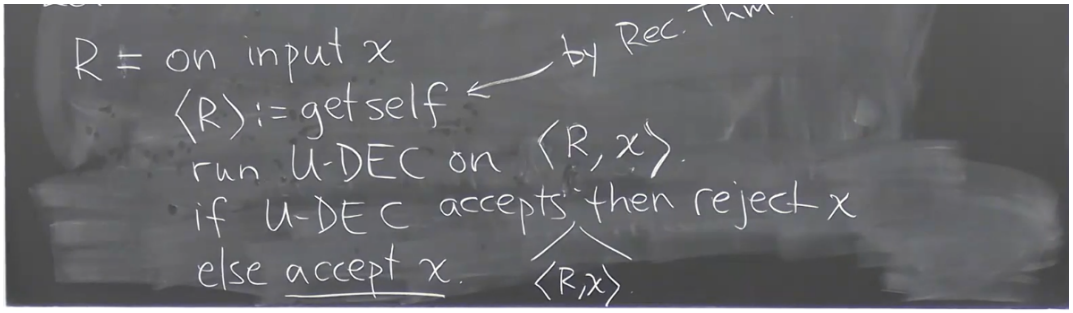


Figure 7: l77

Q: What does R on $\langle R \rangle$ do?

If R accepts input $\langle R \rangle$, then U-DEC did not accept input $\langle R, \langle R \rangle \rangle \implies R$ does not accept $\langle R \rangle$

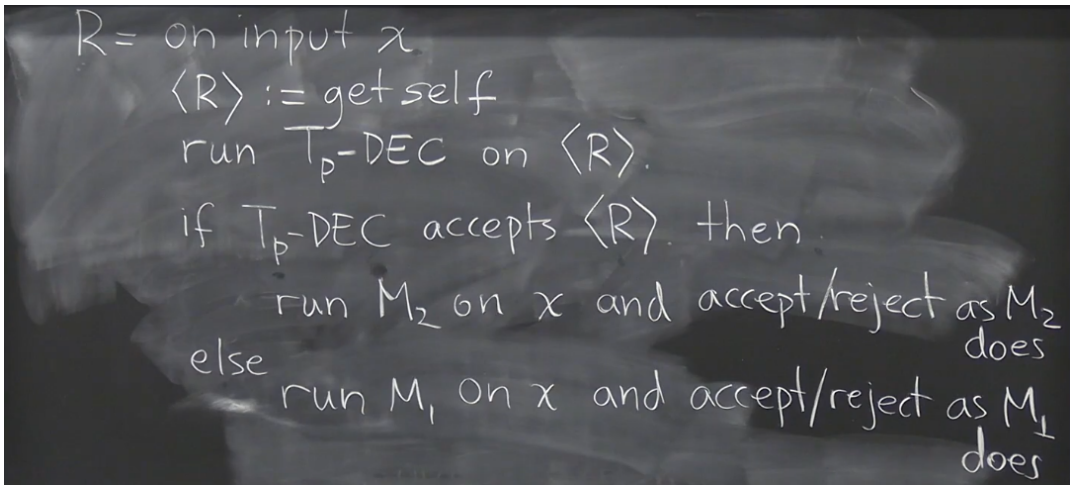
Rice's Thm (via RT):

Let P be non-trivial property of recognizable languages.

$\implies \exists M_1$ s.t. $L(M_1) \in P$ and $\exists M_2$ s.t. $L(M_2) \notin P$

$T_P = \{\langle M \rangle : L(M) \in P\}$ is undecidable

Suppose for contra, that T_P is decidable. Let T_P-DEC be a decider for T_P



Q: $\langle R \rangle \in T_P$?

If yes, R has property P or $L(R) \in P$. Then $R = M_2 \implies L(R) \notin P$

If no, $L(R) \notin P$. Then $L(R) = L(M_1) \in P$

Minimal TM:

TM M is minimal if $\forall$ TMs M' s.t. M' is equivalent to M $[L(M) = L(M')]$, $|\langle M \rangle| \leq |\langle M' \rangle|$ [description of the TM is minimal]

$MIN = \{\langle M \rangle : M \text{ is minimal}\}$

Thm 6.4: MIN is unrecognizable

Proof:

Suppose for contradiction that MIN is recognizable

Let E_{MIN} be an enumerator for MIN

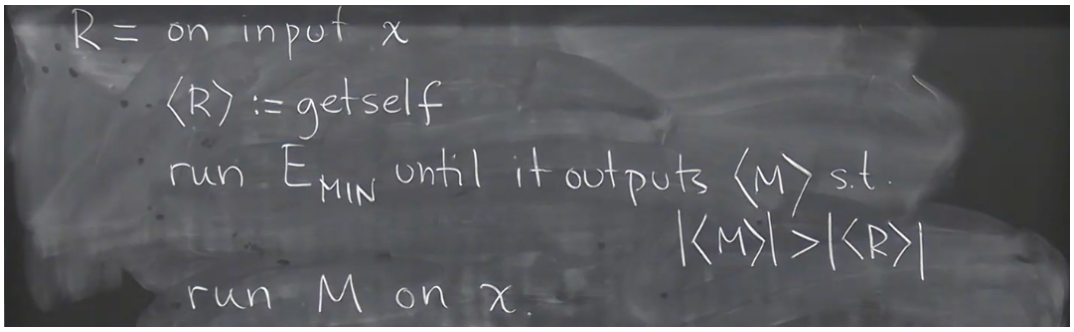


Figure 8: l79

Then R accepts, rejects, loops on the same inputs as M.

$\implies L(R) = L(M) \implies R$ equivalent to M but $|\langle R \rangle| < |\langle M \rangle|$

Therefore E_{MIN} does not enumerate the minimal TMs

3.4 Proof of RT

Lets revisit P1:

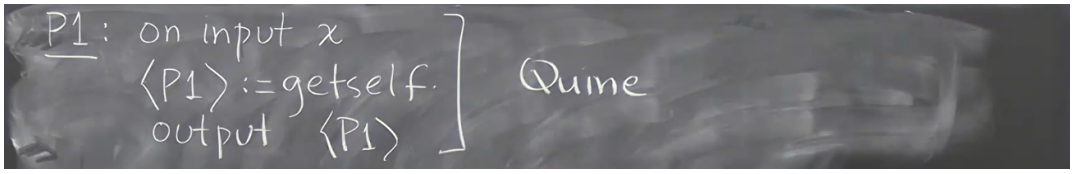


Figure 9: l710

Want TM R that outputs $\langle R \rangle$

$R = AB$, R consists of two TMs working together (A produces some input that B looks at) to produce $\langle AB \rangle$

Lemma: There is a computable function $q : \Sigma^\star \to \Sigma^\star$ s.t. $q(x) = P_x$, program that outputs x $[P_x = \text{output } x]$ -> [let func = (x) => print(x)]

Consider

$R = AB$

$A = P_{\langle B \rangle}$ [A outputs $\langle B \rangle$]

B looks at its input x [expected to be $\langle B \rangle$]

It can deduce what A is: P_x

It can construct $\langle A \rangle$ and $\langle B \rangle$ (input)

$\implies$ can construct contatenation $\langle AB \rangle$

Outputs $\langle AB \rangle = \langle R \rangle$

Note that:

A = output "on input x, $a \leftarrow P_x$; output a, x"

B = on input x, $a \leftarrow P_x$ (can do this by lemma); output a, x

$a \leftarrow$ output "on input x, ..." (instruction of A)

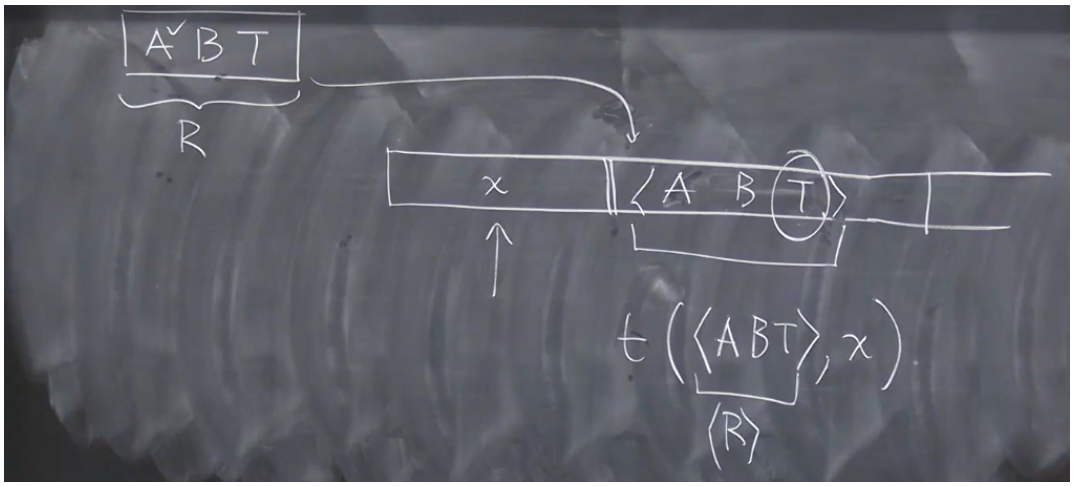
Then B outputs (taking what A outputs as input)

output "on input x, ..." $\circ$ on input x, ...

Part 2:

Now consider $R = ABT$, where A is $P_{\langle BT \rangle}$, B looks at input y [expected to be $\langle BT \rangle$] and deduces $\langle A \rangle$ from $\langle BT \rangle$ on its tape ($A = P_{\langle BT \rangle}$) and outputs $\langle ABT \rangle$ on tape. Finally, use T to compute $t(\langle ABT \rangle, x)$.

Here is a visualization



Basically, we first have x on the tape, then we run A to get $\langle BT \rangle$ as input to B. B takes $\langle BT \rangle$ as input and attempt to deduce $A = P_{\langle BT \rangle}$. Finally, B outputs everything, $\langle ABT \rangle$ onto our original tape. At this point, we will have T to actually compute $t(\langle ABT \rangle, x)$ and we have the description of $R = ABT$.

$\implies$ We can justify using **getself()**