Non-Deterministic T.M. (NTM)

Recall in D.T.M.'s $s(q_i, q_j)$ is either unique or undefined.

N.T.M.'s $s(q_i, q_j)$ is a finite set of the type

$$\{ (p_1, b_1, M_1), \ldots, (p_R, b_R, M_R) \}$$

And any of these options can be used when performing the transition.

As before, string $w$ is accepted if N.T.M. has at least one execution leading to a final state.

Example

$$\Sigma = \{0, 1, 2, 3, \ldots, 9\}$$

$$L = \{ w \in \Sigma^* \mid \text{a zero appears} \ i \ \text{positions to left of some} \ j \ \text{in} \ w, \ \text{with} \ i > 0 \}$$

$$= \{ w \mid \exists j > 0, w_{j-i} = 0 \}$$

Thus $03156 \notin L$

$3720654432 \in L$

NTM $N$

$$Q = \{ q_0, q_f, [p, 0], [p, 1], \ldots, [p, 9] \}$$

$$F = \{ q_f \} \quad \Pi = \{ B, 0, 1, 2, \ldots, 9 \}$$

Idea: scan $w$ left-to-right, guess at some $w_j = i$, store $i$ in CPU register and move $i$ steps left to find 0.
Transitions

\[ \begin{align*}
\cdot & \quad S(q_0, o) = \{(q_0, o, R)\} \quad (\text{since } \omega_j > 0) \\
\cdot & \quad \forall i \geq 0 \\
& \quad S(q_0, i) = \{(q_0, i, R), ([p, i], i, L)\} \\
& \quad \uparrow \\
& \quad \text{GUESSING CURRENT CELL IS } \omega_j \\
\cdot & \quad \forall i > 1, \forall x \in \Gamma \\
& \quad S([p, i], x) = \{(p, i-1], x, L)\} \\
\cdot & \quad \text{ACCEPT} \\
& \quad S([p, 1], o) = \{(0, o, R)\}
\end{align*} \]

Execution Trace

INPUT \( \omega = 103332 \)

\[
q_0 \xrightarrow{1} 19q_0 03332 \xrightarrow{1} 10q_3 3332 \\
\xrightarrow{1} 103 q_3 332 \xrightarrow{1} 10 [p, 3] 3332 \\
\xrightarrow{1} [p, 2] 03332 \xrightarrow{1} [p, 1] 103332 \\
\text{(REJECT)}
\]

\[
q_0 \xrightarrow{1} 103332 \xrightarrow{1} 103 q_0 332 \xrightarrow{1} 103 q_0 332 \\
\xrightarrow{1} 103 [p, 3] 332 \xrightarrow{1} 10 [p, 2] 3332 \\
\xrightarrow{1} [p, 1] 03332 \xrightarrow{1} 10 p 3332 \\
\text{(ACCEPT)}
\]
**Theorem**  
If $N$ is an NTM, then there exists a DTM $D$ such that $L(D) = L(N)$.

**Proof**  
Given $N$ and input $w$, we will show how a multi-tape DTM can simulate $N$'s execution on input $w$. Clearly, we can convert $D$ to a single-tape later.

**Simulation Idea**  
Consider the execution tree of $N$ on $w$.

```
I_0 = \gamma_0 w
\begin{array}{c}
  \text{ID}_0 \\
  \downarrow \\
  \text{ID}_1 \\
  \downarrow \\
  \text{ID}_2 \\
  \downarrow \\
  \text{ID}_3 \\
  \downarrow \\
  \text{ID}_{i1} \\
  \downarrow \\
  \text{ID}_{i2} \\
  \downarrow \\
  \ldots
\end{array}
```

DTM $D$ will perform a BFS of this execution, systematically enumerating the ID's until it finds an accepting ID.

**Two Tapes**  

**Tape 1** holds a "queue" of ID's of $N$ in BFS order.

**Tape 2** scratch tape.
**Evolution of Tape 1's Contents.**

Initially

\[
\ast I D_0 \ast
\]

\[
\ast I D_0 \ast I D_0 \ast I D_0 \ast I D_0 \ast
\]

\[
\ast I D_0 \ast I D_1 \ast I D_2 \ast I D_3 \ast
\]

\[
\ast I D_0 \ast I D_1 \ast I D_2 \ast I D_3 \ast
\]

\[
\ast I D_0 \ast I D_1 \ast I D_2 \ast I D_3 \ast I D_1 \ast I D_1 \ast
\]

\[
\ast I D_0 \ast I D_1 \ast I D_2 \ast I D_3 \ast I D_1 \ast I D_2 \ast I D_2 \ast
\]

\[
\ast I D_0 \ast I D_1 \ast I D_2 \ast I D_3 \ast I D_1 \ast I D_2 \ast I D_2 \ast I D_2 \ast
\]

\[
\ast I D_0 \ast I D_1 \ast I D_2 \ast I D_3 \ast I D_1 \ast I D_2 \ast I D_2 \ast I D_2 \ast I D_2 \ast
\]

\[
\vdots
\]

**Algorithm for D.**

- **Tape 1** contains sequence of ID's in BFS order separated by \(\ast\)'s.

- \(\ast\) marks next ID to be explored — all to left of this have been explored already, and all to right have yet to be explored.

- Initially only I D_0 is \(\ast\) on the tape.

- Use multiple tracks to store \(\ast\) and for other purposes in following algorithm.
Algorithm:

**Step 0**
Examine current IDc (just after *) and read off q1, δ in it.

If q1 ∈ F then accept & halt.

**Step 1**
Let s(q1, δ) have k possible transitions
Copy IDc onto tape 1 & make k new copies of IDc at end of tape 2.

**Step 2**
Modify k copies of IDc on tape 2 to the k possible outcomes of s(q1, δ) being applied

**Step 3**
Move * down past IDc
Clean up tape 2
Return to Step 0

Verify
- Above can be implemented in a DTM
- Correct in that D accepts w if and only if some execution path of N on w will accept w.

Recall
T.M. M (DTM or NTM) is said to have run in time T(n) if on input w with |w| = n all possible execution paths lead to halting in ≤ T(n) transitions.
**Simulation Time**

- **Suppose** NTM N has running time \( T(n) \).
- **Question** What is D's running time?

**Let** \( m \) be the maximum size of \( s(q, a) \) in \( N \)—maximum number of non-deter. choices.

For \( N \) on \( w \) let \( t = T(1|w|) \).

Clearly execution tree of \( N \) on \( w \) has at most \( t \) levels.

Tree size \( S \leq 1 + m + m^2 + \ldots + m^t \)

\[ S = \frac{m^{t+1} - 1}{m - 1} = O(m^t). \]

Thus D has at most \( O(m^t) \) iterations.

**Observe** while DPDA's could not simulate NPDA's, DTMs can simulate NTMs even though NTMs are more powerful than DTMs.

**Why?** Because the very fact that TMs are more powerful gives them the ability to simulate non-determinism—albeit at a terrible increase in running time.
**Algorithms vs Procedures.**

**Algorithms** halting T.M. — always halts in finite time on all inputs and execution paths, regardless of acceptance or rejections.

**Recursive** class of languages accepted by algorithms (also called decidable)

**Undecidable** languages which are non-recursive or don't have algorithms.

**Procedures** arbitrary T.M. (may not halt on w & L)

**Recursively Enumerably (R.E.)** language class defined by T.M. or procedures.

**Remark** assume DTMs in above definitions, but doesn't really matter as NTM's can be simulated.

**Pictures**

**Algorithm**

\[ \text{input } w \rightarrow A \rightarrow \text{YES (w & L)} \rightarrow \text{NO (w & L)} \]

**Procedure**

\[ \text{input } w \rightarrow P \rightarrow \text{YES (w & L)} \]
WORLD OF LANGUAGES

L_u / L_d WILL BE DEFINED SHORTLY.

CLOSURE PROPERTIES

THEOREM
L RECURSIVE $\Rightarrow$ L RECURSIVE

PROOF
GIVEN ALGO A FOR L
CONSTRUCT ALGO $\overline{A}$ FOR $\overline{L}$

ALGO $\overline{A}$

INPUT $\overline{w}$

MEANING (OF PICTURE)

YES $\rightarrow$ NO

$+$ A REF, \[ \begin{cases} \text{REPLACE } s(q, x) = (\theta, -) \text{ by } s(q, x) \text{ UNDEFINE} \\ \text{NO} \rightarrow \text{YES } \text{IF } s(q, x) \text{ UNDEFINED} \text{ THEN SET } s'(q, x, \overline{R}) \end{cases} \]
THEOREM  BOTH $L_1$, $L_2$ ARE R.E. $\implies$ BOTH $L_1^c$, $L_2^c$ ARE REC.

PROOF  LET $P_1, P_2$ BE PROCEDURES FOR $L_1, L_2$

RUN IN PARALLEL TO GET ALGO. $A$

\[
\text{INPUT } w \rightarrow \begin{cases}
  P & \text{YES} \rightarrow \text{YES} \\
  \overline{P} & \text{YES} \rightarrow \text{NO}
\end{cases}
\]

\[\text{ALGO } A\]

FOR ANY $w$ ONE OF $P_1, P_2$ WILL HALT AND GIVE THE RIGHT ANSWER

IMPLEMENTATION?

IN A USE 2 TAPEs, ONE EACH FOR $P$ AND $\overline{P}$

STATES $p$ IN $P$, $q$ IN $\overline{P} \implies \langle p, q \rangle$ IN $A$.

TRANSITIONS \[
\begin{cases}
  s(p, x) = (p', x', M_1) \text{ IN } P \\
  s(q, y) = (q', y', M_2) \text{ IN } \overline{P}
\end{cases}
\]

$\implies s(\langle p, q \rangle, x, y) = (\langle p', q' \rangle, (x', M_1), (y', M_2))$

FINAL? ANY $\langle p, q \rangle$ IN $A$ WHERE $p$ IS FINAL IN $P$

NOTE IF $P$ ACCEPTS, $q$ IS FINAL SO $p$ CANNOT BE FINAL, SO $A$ WILL REJECT.

DONE
Any language \( L \) is such that:

- Either both \( L \) and \( \overline{L} \) are RE.
- Or at least one of \( L, \overline{L} \) is non-RE.

**Possibilities**

<table>
<thead>
<tr>
<th></th>
<th>( L ) RE</th>
<th>( L ) RE but not RE</th>
<th>( L ) non-RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L ) RE</td>
<td>( \checkmark )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( L ) RE but not RE</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( L ) non-RE</td>
<td>( \times )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
</tbody>
</table>

**Theorem**

\( L, L_2 \) RE \( \Rightarrow \) \( L \cup L_2 \) RE.

**Proof**

Assume algorithms \( A_1 \) for \( L_1 \) and \( A_2 \) for \( L_2 \).

Combining \( A_1 \) and \( A_2 \) in series?

If \( A_1 \) halts in non-final state \((w \notin L_1)\)

Then have transition to initial state of \( A_2 \) (having saved \( w \) on second tape).
Theorem: \( L_1, L_2 \) are R.E. \( \Rightarrow \) \( L_1 \cup L_2 \) are R.E.

Proof: Assume procedures \( P_1, P_2 \) for \( L_1, L_2 \)

- Input \( \omega \) to \( P_1 \) and \( P_2 \)
- If \( \omega \in L_1 \cup L_2 \), one of \( P_1 \) and \( P_2 \) will halt and say "yes".

Exercise: Check closure under \( \cap \), reversal.

Question: How do we show that a language is undecidable or non-R.E.?

Key Concept: Universal T.M. (UTM)

\[
\text{UTM} \quad \text{Input} \quad \left\{ \begin{array}{l}
\text{T.M. M} \\
\text{M's input } \omega
\end{array} \right.
\]
\[
\text{Output} \quad \text{Accepts } \langle M, \omega \rangle \iff M \text{ accepts } \omega.
\]

Intuition: \( \text{UTM} = \text{Computer} \)
\( \text{TM} \equiv \text{Program} \).

Thus, need "programming language" to be able to specify T.M. M to UTM.
ENCODING T.M. AS INTEGERS

Consider any T.M. $M = (Q, \Sigma, \Pi, \delta, q_0, B, F)$

Assume
- $\Sigma = \{0,1\}$
- $\Pi = \{0,1,B\}$
- $Q = \{q_1, q_2, ..., q_n\}$
  - Initial state $q_1$
  - Final state $q_2$

Set
- $x_1 = 0, x_2 = 1, x_3 = B$
- $d_1 = \text{LEFT}, d_2 = \text{RIGHT}$

Consider transition $s(q_i, x_j) = (q_k, x_l, d_m)$

With
- $i, k \in \{1, ..., n\}$
- $j, l \in \{1, 2, 3\}$
- $m \in \{1, 2\}$

Encode as bit string $0^i10^j10^k10^l10^m$.

Encoding T.M.

$\langle M \rangle = C_1 \ 11 \ C_2 \ 11 \ C_3 \ 11 \ ... \ 11 \ C_n$

Where $C_1, C_2, ..., C_n$ are the encodings of the $n$ transitions in $M$.

Note we use $\langle M \rangle$ to denote the string encoding T.M. $M$ and to avoid confusion with $M$. 
NOTE TO ENCODE T.M. M ALONG WITH ITS INPUT w

\[ \langle M, w \rangle = \langle M \rangle \text{III} w \]

OBSERVE ANY BIT-STRING WHICH IS A VALID ENCODING WILL ENCODE A UNIQUE T.M. — BUT IF IT IS NOT A VALID ENCODING, WE WILL ASSUME THAT IT ENCODES THE PARTICULAR MACHINE \( M_0 \) WITH 1 STATE AND NO TRANSITIONS

\( (L(M_0) = \emptyset) \).

THUS EACH T.M. M CAN BE ENCODED AS BINARY STRING \( \langle M \rangle \) SUCH THAT:

a) EACH T.M. HAS \( \geq 1 \) ENCODING
b) EACH NUMBER ENCODES A UNIQUE T.M.

ENUMERATING BINARY STRINGS.

WE DEFINE A SPECIFIC ORDERING OF ALL BINARY STRINGS — IN INCREASING ORDER OF LENGTH, AND WITHIN ALL STRINGS OF SAME LENGTH WE ORDER THEM LEXICOGRAPHICALLY

\[ \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots \]

LET \( w_i \) BE THE \( i^{\text{th}} \) STRING IN THIS ORDER \( (w_1 = \varepsilon) \).

DEFINE \( M_i \) AS THE T.M. ENCODED BY \( w_i \).

THUS WE GET AN ORDERING OF T.M.'S IN WHICH EACH T.M. APPEARS AT LEAST ONCE, BUT POSSIBLY MANY TIMES.
The Diagonalization Language

Consider infinite table \( T \) such that \( \forall i, j \in \mathbb{N} \):

\[
T(i, j) = \begin{cases} 
1 & w_j \in L(M_i) \\
0 & w_j \notin L(M_i)
\end{cases}
\]

<table>
<thead>
<tr>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( w_4 )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( M_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( M_4 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus each row is a characteristic vector of \( L(M_i) \), encoding which strings belong to it.

Note we can postulate this table only because we showed ordering enumeration of \( w_i | M \).

Defn Diagonal Language

\[
L_d = \{ w_i \mid T(i, i) = 0 \}
\]

\[
= \{ w_i \mid w_i \notin L(M_i) \}
\]

Thus \( L_d \) is defined by taking diagonal of \( T \), complementing each bit to obtain a language characteristic vector.
**Question** Is \( L_d \) R.E.? Is \( L_d \) REC?

**Theorem** \( L_d \) is non-R.E.

**Proof** Assume \( L_d \) is R.E. and has a T.M.

Then \( \exists k \in \mathbb{N} \) such that \( L(M_k) = L_d \)

**Question** Is \( \omega_k \in L_d \)?

**Case 1** \([\omega_k \in L_d]\)

\[ \Rightarrow \omega_k \in L(M_k) \]
\[ \Rightarrow T(k, k) = 1 \]
\[ \Rightarrow \omega_k \notin L_d \]

**Contradiction**

**Case 2** \([\omega_k \notin L_d]\)

\[ \Rightarrow \omega_k \notin L(M_k) \]
\[ \Rightarrow T(k, k) = 0 \]
\[ \Rightarrow \omega_k \in L_d \]

**Contradiction**

**Intuition** \( L_d \) was defined so that it disagreed with each \( M_k \) on at least string \( (\omega_i) \) — thus none of the \( M_k \)'s could have \( L_c \) as its language

But all T.M. 'M are \( M_k \) for some \( i \), so no T.M. can accept \( L_d \)

**Problem** \( L_d \) is non-constructive.
UNIVERSAL T.M.

**UTM**

**INPUT** \( \langle M, w \rangle \) — ENCODING OF SOME T.M. M AND ITS INPUT W

**ACTION** SIMULATE M ON W, AND ACCEPT \( \langle M, w \rangle \)

IF AND ONLY IF M ACCEPTS W.

---

**QUESTION** WHAT IS ITS LANGUAGE?

**DEFN** UNIVERSAL LANGUAGE \( L_u \)

\[ L_u = \{ \langle M, w \rangle \mid w \in L(M) \} \]

**THEOREM** \( L_u \) IS R.E.

**WHY?** BECAUSE WE CAN CONSTRUCT A UTM U FOR \( L_u \)

---

**UTM U HAS 4 TAPES**

**TAPE 1** CONTAINS \( \langle M, w \rangle \) (READ-ONLY)

**TAPE 2** SIMULATES M'S TAPE

**TAPE 3** CURRENT STATE \( q_i \) OF M

**TAPE 4** SCRATCH TAPE.

---

**EXECUTION**? STEP-BY-STEP SIMULATION OF M ON W

INITIALLY COPY W TO TAPE 2, \( q_i \) TO TAPE 3

EACH STEP USE TAPE 2/3 TO DETERMINE WHICH TRANSITION, AND MODIFY TAPES 2/3 APPROPRIATELY — ACCEPTING IF FINAL ...
Undecidability of $L_U$.

Observe $L_U$ is r.e. since $L_U = L(U)$, but now we show that it isn't recursive.

To do so, we will employ notion of a "reduction" which we have seen earlier informally.

Defn $L_1$ reduces $L_2$ (denoted $L_1 < L_2$) if there exists a function $\theta$ (called the reduction) such that:

1. Some T.M. $M_\theta$ computes $\theta$ by taking as input a string $w$ and halting with string $\theta(w)$ on its tape.

2. $\theta$ is such that $w \in L_1 \iff \theta(w) \in L_2$.

That is, $\theta$ maps all strings in $L_1$ to a subset of strings in $L_2$, and all strings in $\overline{L}_1$ to a subset of strings in $\overline{L}_2$.
**Theorem** \( L_1 \subset L_2 \text{ and } L_2 \text{ is recursive/decidable} \Rightarrow L_1 \text{ is recursive/decidable} \)

**Proof** \( \begin{aligned} \text{Gwen} & \quad \{ \text{Algorithm } A_2 \text{ for } L_2 \\ \text{Algorithm } M_0 \text{ for } \emptyset \} \\ \text{Construct Algorithm } A_1 \text{ for } L_1 \end{aligned} \)

\[ \text{Algo } A_1 \quad \begin{array}{c} \omega \rightarrow M_0 \rightarrow \beta(\omega) \rightarrow A_2 \quad \text{YES} \\ \text{NO} \end{array} \]

\( A_1 \text{ is correct since } \omega \in L_1 \iff \beta(\omega) \in L_2 \quad \text{DONE} \)

**Corollary:** \( L_1 \subset L_2 \text{ and } L_1 \text{ is undecidable/non-rec} \Rightarrow L_2 \text{ is undecidable/non-rec} \)

**Observe**
- Switch in direction of statement
- Can be used to show languages are undecidable

**Remark** Above results apply to R.E. also

**Thus**
- If \( L_1 \subset L_2 \text{ and } L_2 \text{ R.E.} \), then \( L_1 \text{ R.E.} \)

**And**
- If \( L_1 \subset L_2 \text{ and } L_1 \text{ non-R.E.} \), then \( L_2 \text{ non-R.E.} \)
Theorem: \( L_u \) is non-recursvie.

Proof: Since \( L_d \) is non-re.e., follows that \( \overline{L_d} \) is non-rec.

Will show \( \overline{L_d} \) reduces to \( L_u \) \((\overline{L_d} \subset L_u)\)

Thus \( L_u \) must be non-rec.

Recall:

\[ L_d = \{ w_i \mid w_i \notin L(M_i) \} \]

\[ \Rightarrow \overline{L_d} = \{ w_i \mid w_i \in L(M_i) \} \]

Note by defn, \( w_i = \langle M_i \rangle \)

Thus \( \overline{L_d} \) is set of strings accepted by T.M. \( M \) which they encode.

Reduction:

Given string \( w \) which is input for \( \overline{L_d} \), produce string \( \langle M_i, w \rangle \) — input for \( L_u \).

Define \( \theta (w) = \langle w, w \rangle = w \upharpoonright w \)

Clearly \( \uparrow \) halting T.M. \( M \) to convert \( w \) into \( \langle w, w \rangle \).

Need to show:

\[ w \in \overline{L_d} \iff \theta (w) \in L_u . \]

Observe:

\[ w_i \in \overline{L_d} \iff w_i \in L(M_i) \]

\[ \iff \langle M_i, w_i \rangle \in L_u \]

\[ \iff \langle w, w_i \rangle \in L_u \]

\[ \iff \theta (w_i) \in L_u \]

Done.
Thus

- $L_d$ is non-R.E. **Diagonalization Proof**
- $L_d$ is R.E. but non-REC. **Simple Exercise**
- $L_u$ is R.E. but non-REC. **By Reduction from $L_d$**
- $L_u$ is non-R.E. **By Inference from above.**

Observe $L_d$ was shown non-R.E. via a direct proof much like the "Hello-world" proof — since $\forall c L_d$ when $M_i$ doesn't accept its own encoding.

But from now on we can establish new problems are undecidable via reductions, as we did for $L_u$.

**Idea**

A. To show $L$ is non-R.E., give a reduction from known non-R.E. language such as $L_d$ or $L_u$

B. To show $L$ is non-REC, give a reduction from known non-REC language such as $L_d$ or $L_u$.

Note while reductions from $L_d$ or $L_u$ will show that language $L$ is non-REC too, since it also shows $L$ is non-R.E. It cannot work when $L$ is non-REC but R.E.
Consider the following languages:

\[ L_e = \{ <M> \mid L(M) = \emptyset \} \]
\[ L_{ne} = \{ <M> \mid L(M) \neq \emptyset \} \]

Here, \( L_e \) is set of all strings \( w \) such that T.M. \( M \) encoded by \( w \) has an empty language, and \( L_{ne} \) is the complement language.

**Claim 1** \( L_{ne} \) is RE.

**Idea** Construct NTM \( N \) for \( L_{ne} \)
- Given input \( <M> \), \( N \) behaves as follows:
  1) Guess a string \( w \in \Sigma^* \)
  2) Simulate \( M \) on \( w \) (like a UTM)
  3) Accept \( <M> \) if \( M \) accepts on \( w \)

**Clearly** \( <M> \in L(N) \iff \forall w, w \in L(M) \)

**Claim 2** \( L_{ne} \) is non-RE.

**Proof** Give a reduction from \( L_u \) to \( L_{ne} \)

\[ \Theta \quad \text{INPUT} \quad <M, w> \quad \text{INSTANCE OF } L_u \]
\[ \quad \text{OUTPUT} \quad <M'> \quad \text{INSTANCE OF } L_{ne} \]

such that

\[ <M, w> \in L_u \iff <M'> \in L_{ne} \]

\( \Theta \) computable by halting T.M.
DESCRIPTION OF M

Suppose M has input z.

Then M ignores z completely and instead simulates M on w (using UTM).

If M accepts w then M' halts and accepts z.

Otherwise, if M never halts on w or rejects w, M' also never halts or rejects z.

Observe:

\[
\begin{align*}
&\text{if } w \in L(M) \quad L(M') = \Sigma^* \\
&\text{if } w \notin L(M) \quad L(M') = \emptyset
\end{align*}
\]

Thus \( <M,w> \in Lu \iff <M'> \in Lne. \)

Also note it is possible to define halting T.M. M_0 which given M, w can construct M' which behaves as above.

Thus \( Lu < Lne, \) and since Lu is non-REC it follows that Lne is non-REC.

\[ Lne \text{ R.E. but not-REC.} \]

\[ Le \text{ must then be non-R.E.} \]
Observe we showed that a specific property of T.M.
languages (non-emptiness) is undecidable.

Turns out similar proof shows the undecidability of
all non-trivial properties of R.E. languages.

**Property** \( P \) can be represented by a language
\[
L_P = \{ <M> \mid L(M) \text{ has property } P \}
\]

Observe \( P \) is not a language property of T.M.'s
(e.g., has \( \leq 100 \) states), but of their languages
— thus all T.M.'s with the same language are
either all in \( L_P \) or all not in \( L_P \)

Non-trivial property at least one R.E. language has
property \( P \), and at least one doesn't.

**Rice's Theorem** Every non-trivial property of R.E.
languages is undecidable

**Proof** Fix any non-trivial property \( P \)

Assume \( \varnothing \) doesn't have property \( P \)

(otherwise work with \( \overline{P} \) — since \( P \) is
decidable if and only if \( \overline{P} \) is decidable)

will show \( L_{\varnothing} \subset L_P \).
Since \( P \) is non-trivial, can assume there is some \( M_1 \) such that \( L(M_1) \) has property \( P \) \((M_1 \in L_P)\) and we already assumed \( \emptyset \) doesn't have \( P \).

**Idea**
Reduction \( \phi \) takes as input \( \langle M, w \rangle \) an instance of \( L_u \) and produces \( M_2 \).

\( M_2 \)'s behavior

- **Ignores** its own input \( z \) at first
- **Simulates** \( M \) on \( w \)
- **If** \( M \) accepts \( w \), \( M_2 \) simulates \( M_1 \) on \( z \) and accepts if \( M_1 \) accepts \( z \)
- **If** \( M \) rejects \( w \) (or doesn't halt)

\( M_2 \) does the same

Thus

- \( w \in L(M) \Rightarrow L(M_2) = L(M_1) \) [has \( P \)]
- \( w \notin L(M) \Rightarrow L(M_2) = \emptyset \) [hasn't \( P \)]

or \( \langle M, w \rangle \in L_u \iff M_2 \in L_P \)

**Follows** \( L_P \) is undecidable

**Done**

Note: The proof was same, with \( L(M_1) = \Sigma^* \).