Beyond RE and co-RE

We saw (by a counting argument) that there is some language that is neither RE or co-RE. We will prove this for a natural language: \( \text{EQ}_{\text{TM}} = \{<M_1, M_2> : L(M_1) = L(M_2)\} \)

Recall:
- \( A_{\text{TM}} \) is undecidable, but RE
- co-\( A_{\text{TM}} \) is undecidable, but coRE

Therefore, not in co-RE
Therefore, not in RE

Theorem: \( \text{EQ}_{\text{TM}} \) is neither RE nor coRE.

Proof:
- not RE:
  - reduce from co-\( A_{\text{TM}} \) (i.e. show co-\( A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}} \))
  - what should \( f(<M, w>) \) produce?
- not co-RE:
  - reduce from \( A_{\text{TM}} \) (i.e. show \( A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}} \))
  - what should \( f(<M, w>) \) produce?

Proof: \( A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}} \)

\[
\begin{align*}
\text{TM } M_1: & \text{ on input } x, \\
& \text{ accept if } M \text{ accepts } w \\
\text{TM } M_2: & \text{ on input } x, \\
& \text{ simulate } M \text{ on input } w \\
& \text{ accept if } M \text{ accepts } w
\end{align*}
\]

\[f(<M, w>) \in \text{EQ}_{\text{TM}} \text{ if and only if } L(M_1) = \Sigma^* \text{ and } L(M_2) = \Sigma^*\]

Proof: \( \text{co-}A_{\text{TM}} \leq_m \text{EQ}_{\text{TM}} \)

\[
\begin{align*}
\text{TM } M_1: & \text{ on input } x, \\
& \text{ reject if } L(M_1) = \Sigma^* \\
\text{TM } M_2: & \text{ on input } x, \\
& \text{ simulate } M \text{ on input } w \\
& \text{ accept if } M \text{ accepts } w
\end{align*}
\]

\[f(<M, w>) \in \text{EQ}_{\text{TM}} \text{ if and only if } L(M_1) = \emptyset \text{ and } L(M_2) = \Sigma^*\]
Summary

The Recursion Theorem

- A very useful, and non-obvious, capability of Turing Machines:
  – in the course of computation, can print out a description of itself!
- how is this possible?
  – an example of a program that prints out self:
    Print two copies of the following, the 2nd one in quotes:
    “Print two copies of the following, the 2nd one in quotes:”

The Recursion Theorem

- Why is this useful?
- Example: slick proof that $A_{TM}$ undecidable
  – assume TM $M$ decides $A_{TM}$
  – construct machine $M'$ as follows:
    on input $x$,
    • obtain own description $<M'>$
    • run $M$ on input $<M',x>$
    • if $M$ rejects, accept; if $M$ accepts, reject.

The Recursion Theorem

- Lemma: there is a computable function $q: \Sigma^* \rightarrow \Sigma^*$
  such that $q(w)$ is a description of a TM $P_w$ that prints out $w$ and then halts.
- Proof:
  – on input $w$, construct TM $P_w$ that has $w$ hard-coded into it; output $<P_w>$

The Recursion Theorem

- Warm-up: produce a TM SELF that prints out its own description.
- Two parts:
  – Part A:
    • output a description of $B$
    • pass control to $B$.
  – Part B:
    • prepend a description of $A$
    • done
Note: $<A> = q(<B>)$

The Recursion Theorem

Recall: $q(w)$ is a description of a TM $P_w$ that prints out $w$ and then halts.

Part A:
  • output a description of $B$
  • pass control to $B$.
Part B:
  • prepend a description of $A$
  • done
Note: $<A> = q(<B>)$

B
• read contents of tape
• apply $q$ to it
• prepend* result to tape

*combine with description on tape to produce a complete TM
The Recursion Theorem

Theorem: Let T be a TM that computes \( f_n : \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \)

There is a TM R that computes the fn:
\( r : \Sigma^* \rightarrow \Sigma^* \)
defined as \( r(w) = t(w, <R>) \).

- This allows “obtain own description” as valid step in TM program
  - first modify TM so that it takes an additional input (that is own description); use at will

Proof outline: TM R has 3 parts
- Part A: output description of BT
- Part B: prepend description of A
- Part “T”: run TM T

Proof details: TM R has 3 parts

Part A: output description of BT

- \( <A> = q(<BT>) \)

Part B: prepend description of A

- read contents of tape \( <BT> \)
- apply q to it \( q(<BT>) = <A> \)
- prepend to tape \( <ABT> \)

Part “T”: run TM T

- 2nd argument on tape is description of R

Recall: \( q(w) \) is a description of a TM \( P_w \) that prints out \( w \) and then halts.

- watch closely as TM AB runs:
  - A runs. Tape contents: \( <B> \)
  - B runs. Tape contents: \( q(<B>) <B> ) <AB> \)
  - AB is our desired machine SELF.

Note: \( <A> = q(<B>) \)