Theorem.

1. if $A \leq_m B$ and $B$ is r.e then $A$ is r.e.
   ↳ if $A \leq_m B$ and $A$ is not r.e then $B$ is not r.e.

2. if $A \leq_m B$ and $B$ is recursive then $A$ is recursive
   ↳ if $A \leq_m B$ and $A$ is not recursive then $B$ is not recursive.

Proof.

1. Suppose $A \leq_m B$ via $\sigma$ and $B$ is r.e. $B = L(M)$
   Construct a TM $N$ s.t $L(N) = A$ as follows:

On input $x$,
1. Compute $\sigma(x)$
2. Run $M$ on $\sigma(x)$
3. Accept if $M$ accepts.

$N$ accepts $x$ iff $M$ accepts $\sigma(x)$ iff $\underbrace{\sigma(x) \in B \text{ iff } x \in A}$
Definition of $A \leq_m B$.

2. Recall: $A$ is recursive iff $A$ is r.e and $\bar{A}$ is r.e.

Suppose $A \leq_m B$ via $\sigma$ and $B$ is recursive.
Then $\bar{A} \leq_m \bar{B}$ via $\sigma$ [follows from the definition]

if $B$ is recursive then both $B$ and $\bar{B}$ are r.e.

By part 1, both $A$ and $\bar{A}$ are r.e. $\Rightarrow$ $A$ is recursive

Example 1. $FIN = \{M \mid L(M) \text{ is finite}\}$ is not r.e

We give a reduction: $\overline{HP} \leq_m FIN$ ——— (a)

$$\overline{HP} = \{M\#x \mid M \text{ does not halt on } x\}.$$

(a) and Theorem $\Rightarrow$ FIN is not recursively enumerable

$\overline{HP} \leq_m FIN$ : From $M\#x$ construct a TM $M_1 = \sigma(M\#x)$ such that

$M$ does not halt on $x$ iff $L(M_1)$ is finite.
$M_1$ on input $y$ works as follows.

1. Erases the input $y$
2. Writes $x$ on the tape
3. Runs $M$ on input $x$
4. Accept if $M$ halts on $x$.

Descriptions of M and $x$ are hard coded in $M_1$

if $M$ does not halt on $x$ then $M_1$ does not reach step 4
Thus $M_1$ does not accept its input $y$.

$M$ halts on $x \Rightarrow L(M_1) = \Sigma^* \Rightarrow L(M_1)$ is infinite.
$M$ does not halt on $x \Rightarrow L(M_1) = \phi \Rightarrow L(M_1)$ is finite
Thus $\overline{HP} \leq_m FIN$.

$FIN = \{M \mid L(M) \text{ is finite}\}$ is not r.e

$\overline{FIN}$ is not r.e.

$\overline{HP} \leq_m \overline{FIN}$

By definition of $\leq_m$, if $\bar{A} \leq_m \bar{B}$ via $\sigma$ then $A \leq_m B$ via $\sigma$.

Sufices to show that $HP \leq_m FIN$ via some $\tau$

i.e., given $M$ and $x$, construct $M_2 = \tau(M\#x)$ s.t

$M$ halts on $x$ iff $L(M_2)$ is finite.

$M_2$ on input $y$ works as follows:

1. Save $y$ on one of the tracks
2. Write $x$ on a separate track
3. Simulate $M$ on $x$ for $|y|$ steps

$\}$ $M$ and $x$ are hard coded in $M_2$

Erase one symbol in $y$ for each step of $M$ on $x$

4. Accept if $M$ has not halted in $|y|$ steps. otherwise reject

if $M$ does not halt on $x \Rightarrow M_2$ halts and accepts $y$ - $\forall y$.

if $M$ halts on $x$ then it halts after some $n$ steps.

$M_2$ accepts $y$ if $|y| < n$, rejects $y$ if $|y| \geq n$.

$M$ does not halt on $x \Rightarrow L(M_2) = \Sigma^* \Rightarrow L(M_2)$ is infinite

$M$ halts on $x \Rightarrow L(M_2) = \{y \mid |y| < \text{running time of } M \text{ on } x\}$

$\Rightarrow L(M_2)$ is finite

# Rice's Theorem.

**Theorem.** Every non-trivial property of r.e. sets is undecidable.

Property of the r.e sets is a function.

$$P : \{\text{r.e. subsets of } \Sigma^*\} \to \{\top, \bot\}$$

Examples.

Emptiness $\quad P(A) = \begin{cases} \top & \text{if } A = \phi \\ \bot & \text{if } A \neq \phi \end{cases}$

Finiteness $\quad P(A) = \begin{cases} \top & \text{if } A \text{ is finite} \\ \bot & \text{if } A \text{ is not finite} \end{cases}$

Regular $\quad P(A) = \begin{cases} \top & \text{if } A \text{ is regular} \\ \bot & \text{if } A \text{ is not regular} \end{cases}$

Question. For a property $P$ of r.e. sets, is $P$ decidable?

The set has to be given a finite presentation.

Assumption. The r.e. set is presented as a Turing machine whose language is the set.

Note. Property $P$ is that of the set not a property of the Turing machine.

Example. Does a TM $M$ have 100 states?
Does a TM $M$ halt on $\epsilon$ in 100 steps? $\Big\}$ Decidable.

Non-trivial. The property is not universely True or False

That is, there exists r.e sets A & B s.t $P(A) = \top$ and $P(B) = \bot$

Proof of Rice's Theorem. Let P be non-trivial.

Wlog, assume $P(\phi) = \bot$ and $P(A) = \top$ for some A.

Let $L(K) = A$ for some TM K [Note A is r.e]

We give a reduction $HP \leq_m \underbrace{\{ M \mid P(L(M)) = \top \}}$.

Conclude - not recursive.

Given $M$ and $x$, construct $M' = \sigma(M \# x)$.
$M'$ on input $y$ works as follows:

1. Saves $y$ on one of its tracks.

2. Write $x$ on a separate track
3 Runs $M$ on input $x$

$\Big\}$ $M$ and $x$ are hard coded in $M'$.

4. if $M$ halts on $x$, $M'$ runs $K$ on input $y$

$\hookrightarrow L(K) = A$.

$M'$ accepts if $K$ accepts.

if $M$ does not halt on $x$ $\Rightarrow$ step 3 never stops.
$\Rightarrow y \notin L(M')$ for all $y$.

if $M$ halts on $x \Rightarrow$ Step 4 is executed
$\Rightarrow y \in L(M')$ iff $y \in L(K)$.

$M$ halts on $x$ $\Rightarrow L(M') = A \Rightarrow P(L(M')) = P(A) = \top$

$M$ does not halt on $x \Rightarrow L(M') = \phi \Rightarrow P(L(M')) = P(\phi) = \bot$.

$HP \leq_m \{M \mid P(L(M)) = \top\} \Rightarrow$ not recursive

Thus it is undecidable if $L(M)$ satisfies $P$.

# Problems about CFLs

Membership problem. Given a CFG $G$ and a string $x$, is $x \in L(G)$?

Answer. Decidable — CKY algorithm

Emptiness problem. Given a CFG $G$, is $L(G) = \emptyset$?

Answer. Decidable.

# Problems about CFLs

Membership problem. Given a CFG $G$ and a string $x$, is $x \in L(G)$?

Answer. Decidable — CKY algorithm

Emptiness problem. Given a CFG $G$, is $L(G) = \emptyset$?

Answer. Decidable.

Universality problem. Given a CFG $G$, is $L(G) = \Sigma^*$?

Answer. Undecidable

# Valid Computation Histories

## Configurations of a Turing machine

A configuration of a Turing machine $M$ is a triple $(q, y, n)$ where

- $q$ is a state,
- $y$ describes the content of the tape,
- $n$ an integer describing the head position.

Encoding configurations. We can encode configurations as finite strings over the alphabet $\Gamma \times (Q \cup \{-\})$.

Example. Configuration $(q, y, k)$

$$\begin{array}{cccccccc} \vdash & b_1 & b_2 & b_3 & \cdots & b_k & \cdots & b_m \\ - & - & - & - & & q & \cdots & - \end{array}$$

Machine is scanning the $k^{th}$ cell

Start Configuration.

$$\begin{array}{ccccc} \vdash & a_1 & a_2 & \cdots & a_n \\ s & - & - & \cdots & - \end{array}$$

where $x = a_1 a_2 \cdots a_n$

# Valid Computation Histories

Alphabet: $\Gamma \times (Q \cup \{-\})$.

A valid computation history of $M$ on $x$ is a string

$$\#\alpha_0\#\alpha_1\#\alpha_2\#\cdots\#\alpha_N\#$$

- $\alpha_0$ is a start configuration of $M$ on $x$,
- $\alpha_N$ is a halting configuration (state is either the accept state $t$ or reject state $r$),
- $\alpha_{i+1}$ follows in one step from $\alpha_i$ according to $\delta$ of $M$. That is, for $0 \leq i \leq N-1$,

$$\alpha_i \xrightarrow[M]{1} \alpha_{i+1}.$$

Let $\Delta = \{\#\} \cup (\Gamma \times (Q \cup \{-\}))$, then

$$\text{VALCOMPS}(M, x) = \{\text{valid computation histories of } M \text{ on } x\} \subseteq \Delta^*.$$

$$\text{VALCOMPS}(M, x) = \varnothing \text{ iff } M \text{ does not halt on } x.$$

$\overline{\text{VALCOMPS}(M, x)} = \Delta^*$ iff $M$ does not halt on $x$.