Recursion theorem & Hierarchy thm.

* → It's a mathematical result that plays an important role in advanced work in the theory of computability
* → related with self-producing systems. and even computer viruses.
* Consider the paradox
  1. living things are machine
  2. living things can self-produce
  3. machines can not self produce

through biology

In a car factory → More complex
cars are being produced → less complex

If Machine A construct Machine B
Machine A → More complex
Machine B → less complex.

but a machine can not be more complex to itself so making machines that can reproduce themselves is possible.

Self Reference :-

Begin by making a machine that ignores its input and prints out a copy of its own description. We call this m/c SELF

Lemma:- There is a computable function $q: \Sigma^* \to \Sigma^*$, where w is any string. $q(w)$ is the description of a Turing m/c P_w that prints out w and then halts.

$\langle SELF \rangle = \langle A\ B \rangle$

Job of A is to print out a description of B and conversely Job of B is to print out a description of A.

For A we use the m/c $P_{\langle B \rangle}$ described by $q(\langle B \rangle)$. Thus part A is a Turing m/c that prints out $\langle B \rangle$.

If B can obtain $\langle B \rangle$, it can apply q to that and obtain $\langle A \rangle$. But how does B obtain $\langle B \rangle$? It was left on the tape when A finished

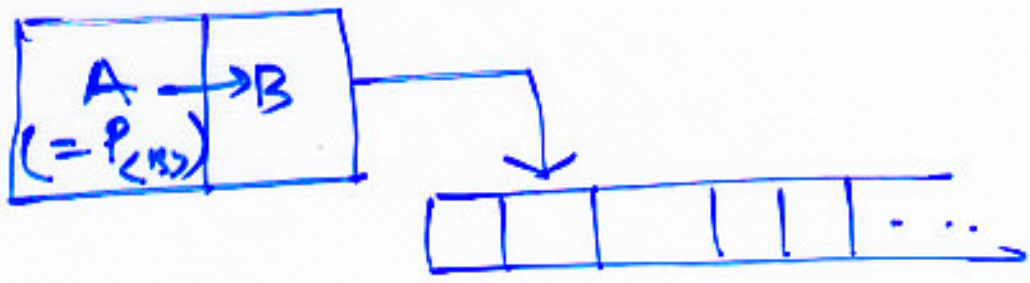
Then after B computes $q(\langle B\rangle) = \langle A\rangle$, It Combines A and B into a single Machine and writes its description on the tape

$\langle AB\rangle = \langle SELF\rangle$

$A = P_{\langle B\rangle}$ and

B = "On input $\langle M\rangle$, where M is a portion of a TM:

1. Compute $q(\langle M\rangle)$
2. combine the result with $\langle M\rangle$ to make a complete TM
3. print the description of this TM and halt



1. First A runs, It prints $\langle B\rangle$ on the tape
2. B starts It looks at the tape and finds its input $\langle B\rangle$
3. B calculates $q(\langle B\rangle) = (\langle A\rangle)$ and Combines that with $\langle B\rangle$ into a TM

Recursion theorem:-

Let T be a TM that computes $t: \Sigma^* \times \Sigma^* \to \Sigma^*$

There is a Turing M/C R that computes $r: \Sigma^* \times \Sigma^* \to \Sigma^*$

where for every w

$$r(w) = t(\langle R \rangle, w)$$

Recursion theorem produces a new M/C R, which operates exactly as T does but with R's description filled automatically.

Hierarchy Theorem.

"More (of the same) resources ⇒ More power"

"Does more space allow for powerful Turing M/C"

"Does more time allow for more powerful Turing M/C".

Ex: Turing M/C should be able to decide more languages in time n^3 than they can in time n^2. The hierarchy theorems prove that this institution is correct subject to certain condition.

These theorems prove that the time and space complexity classes are not same. They form a hierarchy whereby the classes with larger bounds contains more languages than do the classes with smaller bound

if $f(n)$ and $g(n)$ are two space bounds, where $f(n)$ is asymptotically larger than $g(n)$ suppose $f(n)$ exceeds $g(n)$ by only a small and hard to compute amount then the M/C may not be able to use space profitably because even computing the extra space may require more space than available.

A function f is space constructible if there exist a TM M that on unary input n (1^n) output f(n) [ie say binary] and uses O(f(n)) space where f(n) > log n.

Space hierarchy thm :-

for any space constructible fn $f: N \to N$, a language A exists that is decidable in O(f(n)) space but not in o(f(n)) space. proof: on board.

A function $t: N \to N$, where t(n) is at least O(n log n), is called time constructable if the function maps the string 1^n to the binary representation of t(n) is computable in time O(t(n))

Time hierarchy theorem —

for any time constructible function $t: N \to N$, a language A. exists that is decidable in O(t(n)) time but not decidable in time O(t(n)/log(t(n)).

proof: on board.

Church Turing Hypothesis

* → Intuitive notion of algorithms.
= Turing M/c algorithms

* → Any process which could be naturally called an effective procedure can be realized by a Turing M/c