CHURCH TURING THESIS

First clause → Power of turing machines is equal to power of partial recursive functions

We can say that TMs essentially compute partial recursive functions.

What are partial recursive function?

⇓ ('we saw them' in CS201 course)

Finite no. of application of with primitive operations

composition — successor

primitive recursion — zero

bounded / unbounded → μ-recursive predicate — projection

$$f(x_1, x_2, \ldots x_n, 0) = \overset{\text{total func}}{g}(x_1, x_2, \ldots, x_n)$$

$$f(x_1, x_2, \ldots, x_n, y+1) = \underset{\text{total func.}}{h}(x_1, x_2, \ldots, x_n, f(x_1, x_2, \ldots x_n, y))$$

$$\mu z(\overset{\text{total func}}{P}(x_1, \ldots, x_n, z))$$

→ returns first z for which $P(\ )$ is true

unbounded

and if we use μ^x → it is bounded

here μ will search for $z = 0, 1, \ldots, x$.

Every partial recursive function is turing computable

- successor function

lets us have tape in unary representation

| 1 | 1 | 1 | 1 | ⊔ | ⊔ ...

↓

| 1 | 1 | 1 | 1 | 1 | ⊔ | ⊔

- Zero function

| 1 | 1 | 1 | 1 | 1 | ⊔ | ⊔ ...

↓

| 0 | 0 | 0 | 0 | 0 | ⊔ | ⊔ ...

don't argue about representation, see that anything could be made 0.
so zero function is turing computable

- projection function

$\underbrace{| 1 \; 1 \; 1 \; | \#}_{x_1} \; \underbrace{\quad\quad\quad}_{x_2} \; | \# | \; \underbrace{\quad\quad\quad}_{x_3} \; \ldots$

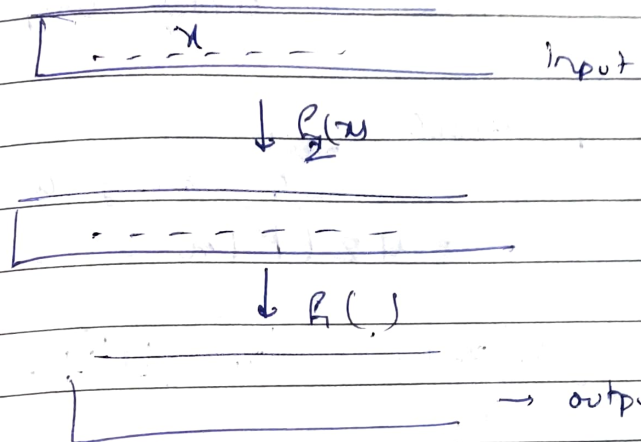
~~Let~~ We are asked $proj_2$

$\underbrace{| \quad\quad\quad}_{x_2} | ⊔ | ⊔$

copy x_2 and blank all after that

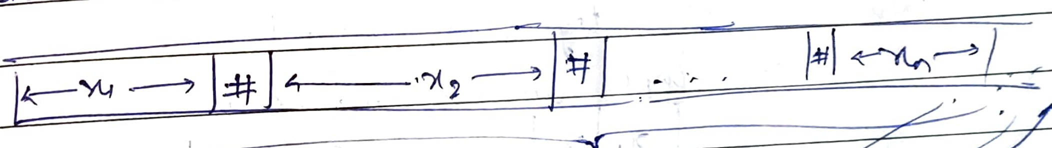
- composition.

$f_1 \circ f_2(x)$ $\quad$ f_1, f_2 are turing computable.



So composition is also turing computable.

- primitive recursive function



duplicate them ahead

To compute this we are doing this process. ← $f(x_1, \ldots x_n, y+1) = h(x_1, \ldots x_n, f(x_1, \ldots x_n, y))$

with the duplicated pair we will find $(f(x_1, \ldots x_n, y))$

⇓

This can indeed be done inductively

like for $f(x_1 \ldots x_n, 2)$

we duplicate $(x_1, \ldots x_n)$

calculate $f(x_1, \ldots x_n, 1)$

but for this we again duplicate to find

So ultimately we get

$f(x_1, \ldots x_n, y+1)$ on tape

which is equal to ← $f(x_1, \ldots x_n, 0)$

$g(x_1, x_2, \ldots x_n)$.

Page No.:
Date:

$f(x_1, x_2, \ldots x_n, 0) = g(x_1, x_2, \ldots x_n)$.

this ~~is~~ can also be easily calculated.

So primitive recursion is turing computable.

- μ- recursive predicate

total → turing computable.

$f(x_1, x_2, \ldots, x_n) = \mu z(P(x_1 \ldots, x_n, z))$

put z = 0 initially

x_1	x_2	..	x_n	0	⊔	⊔	..

↓ return 0 or 1

0 → if false, 1 → if true

0

now we will try with z = 1

so tape contents refreshed

x_1	x_2	---	x_n	1

↓

and so on

as soon as 1 is returned machine halts and returns that z

Note: It may happen that ~~machi~~ for all value of z we get 0, meaning machine infinitely runs. Thus TM is a recogniser and not a decider.

So ~~μ- recu~~ μ-recursive functions are turing computable.

M T W T F S S
Page No.
Date
YOUVA

- Now we will prove
  Every turing computible function is partial recursive.

  For a turing machine M whatever it is computing, it could be simulated using partial recursive function

Godel Numbers

Used to encode multiple numbers into single number.

$$gn_n(x_0, x_1, \ldots, x_n) = \prod_{i=0}^{n} \text{prime number}(i)^{x_i+1}$$

eg. $gn(0, 6, 12) = \underbrace{2^{0+1} \cdot 3^{6+1} \cdot 5^{12+1}}_{}$

$= k$

decoding can be done using μ-recursive predicate. Like

~~decode(i,x) = μz(~~

$$\text{decode}(i, x) = \mu z(\text{compliment}(\text{divides}(x, \text{prime}(i)^{z+1}))) - 1.$$

so let

x be $2^1 3^7 5^{13}$ decode$(1, x) = 6$.

~~decode~~

for $z = 0$

$\frac{2^1 \cdot 3^7 \cdot 5^{13}}{3^1}$

div → true

Comp makes false

for $z = 1$

( )

similar.

for $z = 7$

$\frac{2^1 \cdot 3^7 \cdot 5^{13}}{3^8}$

not divisible

Complement is true.

$\mu z()$ returns $z = 7$.

~~tosi~~ Consider a DTM $(Q, \Sigma, T, \delta, q_0, F)$

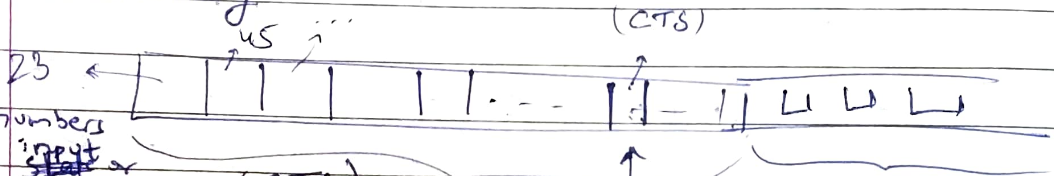
$\Sigma = \{0, 1\}$

Q, T } each state or tape symbol is assigned a unique number.

Configuration (saving state of a machine)
- current state (cs)
- current head pointer ~~(chp)~~ (chp)
- current tape contents (ctc) — just the non blank region.

These 3 things must be saved to have a configuration

Pictorially



These numbers denote ~~state~~ input or tape symbol

(CTC). All these numbers are ~~stored~~ stored in a godel number

chp

no need to store blanks.

symbol at chp = CTS current tape symbol

current state (cs) is also a unique number

Let after the transition current tape contents (CTC) transformed into new tape contents (NTC)

similarly current head pointer (chp) changes to new head pointer (nhp)

current state (cs) $\rightarrow$ new state (ns)

~~We are haved def~~

We are able to define different ~~states~~, configurations as numbers.
Now we will represent transitions as numbers.

$$x = gn_2(cs, chp, ctc)$$

ctc → it itself is a godel number.

x ↓ represents current configuration

Let the transistions be

$$\delta(q, \alpha) = (q', \alpha', L)$$
$$\delta(q, \beta) = (q'', \beta', R)$$

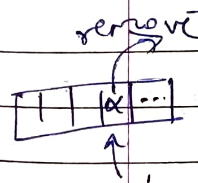
$cs = q$,

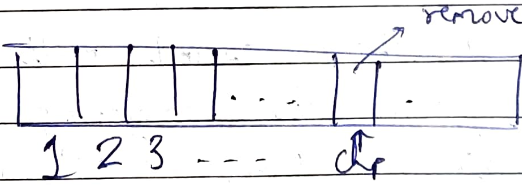
So defining ns,

$$ns = eq(cs, q) \cdot eq(cts, \alpha) \cdot q' + eq(cs, q) \cdot eq(cts, \beta) \cdot q'' + eq(cs, q) \cdot ne(cts, \alpha) \cdot ne(cts, \beta) \cdot cs$$

eq → equal func
ne → not equal
} partial ~~primitive~~ recursive functions (defined in prev sem, CS201)

So ns is a partial recursive function





remove α (cts) and insert α'(nts)

and then compute new godel number = ntc

$$ntc = \left[quotient\left(ctc, prime\ number(chp)^{cts+1}\right) \right] \cdot \left[\left(prime\ number(chp)^{nts+1}\right) \right]$$

multiply symbol

~~ntc~~ can be seen as

$$nhp = eq(cs,q) \cdot eq(cts, \alpha) \cdot (chp - 1) + eq(cs,q) \cdot eq(cts, B) \cdot (chp + 1) + eq(cs,q) \cdot ne(cts,\alpha) \cdot ne(cts,B) \cdot chp$$

ns, ntc, nhp all are partial recursive functions.

x was $gn(cs, chp, ctc)$, godel number representing current state

$$y = gn(ns, nhp, ntc)$$

new state godel number.

So like if $config(0) = gn(0, 0, 2^{1+1} \cdot 3^{1+1} \cdot 5^{0+1} \cdot 7^{1+1} \ldots)$

gn corresponding to this tape contents

1	1	0	1	1	0	⊔	⊔	...

So for some $j+1$ configuration

$$config(j+1) = (ns(config(j)), nhp(config(j)), ntc(config(j)))$$

Halting state $\overset{when}{\Rightarrow}$ $config(k) = config(k+1)$

to check after which transistion TM terminates or it doesn't.

$term(x) = \mu z [eq(config(z), config(z+1))]$

(partial recursive function)

If a machine doesn't halts, then $term(x)$ is not going to return anything.

So $term(x)$ may or may not be a total function

In previous 4 pages, we saw how a turing machine is simulated using partial recursive functions

Turing Machines compute partial recursive functions and hence power of both is same //.