Church: μ-recursive

$0, \sigma, \pi_i^k$ 0, PR, μ-rec] → high-level.

PR ↓ for loop; μ-rec ↓ while loop.

Turing: Turing machine] → low level.

Can we compute every μ-recursive fn on a Turing m/c?

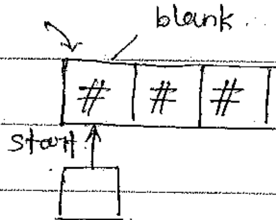
In unary code, 0

1   1

2   11

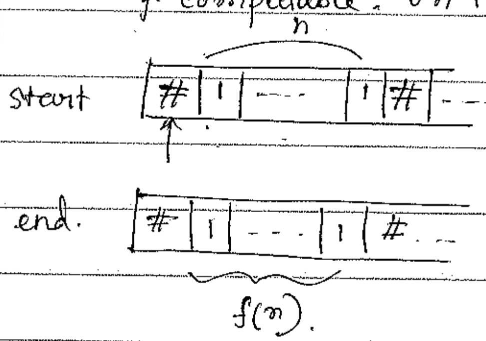
3   111

[There is start state & halt-state in turing m/c.]



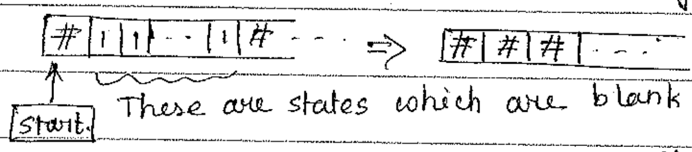
* (here we are building the turing m/c where the pointer doesn't fall off the clip.)

f computable on turing m/c if there exists a turing m/c with the following property.



Starts with value n (state) & end up with state f(n)

how 0 is represented? 0 means nothing on tape



These are states which are blank for '0'

* since the pointer doesn't fall off, when $\frac{\text{State}}{1}$ gets over then rewinds back & erase, & when nothing to erase then halts.

Algorithm :- (start, #) → (R, reading)

(reading, 1) → (R, reading)

(reading, #) → (L, erasing)

(erasing, 1) → (L, #, ")

(erasing, #) → halt

(1) |#|1|1|#| — start
(2) |#|1|1|#| — reading
(3) |#|1|1|#| — reading
(4) |#|1|1|#| — reading
(5) |#|1|1|#| — erasing
(6) |#|1|#|#| — erasing
(7) |#|#|#|#| — erasing
(8) |#| . — halt

when we have '0'

|#|#| start → |#|#| reading → |#|#| erasing → |#|#| halt. So, '0' is turing computable

when 6. (with unary notation : 11 + 1 = 111)

|#|1|1|#| start

(start, #) → (R, reading)
(Reading, 1) → (R, ")
(" , #) → (1, L, back) → here we are adding 1 & going back.
(back, 1) → (L, back)
(" , #) → halt.

here adding 11 & 1.

|#|1|1|#| start → |#|1|1|#| reading → |#|1|1|#| reading → |#|1|1|#| reading

|#|1|1|1| back → |#|1|1|1|#| back → |#|1|1|1|#| back → |#| halt

So, 6 is turing computable.

1. h.w. design a turing m/c. that computes. $f(n) = n+2$.
2. π_1^3.
3. π_2^2 $(n_2, n_2) \to n_2$

If adding with 0 & 1

|#|#| start → |#|#| Reading → |#|1| back → |#|1| halt.

Projection. $\pi_1^2 = (n_1, n_2) \to n_1$

|#|1|1|1|#|1|1|#| (n_1, n_2)
↓
|#|1|1|1|#|#|#|#| (n_1, erase n_2)

(start, #) → (R, ~~reading~~ in1right)
(~~Reading~~ in1right, 1) → (R, ~~reading~~ in1right)
(in1right, #) → (R, in2right)
(in2right, 1) → (R, in2right)
(in2right, #) → (L, erase2)
(in2right, 1) → (#, L, erase2)
(erase2, #) → (L, in1left)

(in1left, 1) → (L, in1left)

(in1left, #) → halt.