Th. No algorithm is possible that given a program p, checks whether p always return ∅.
↖ that always halts.

$$\forall n(p(n)=\emptyset)$$

Th. No algorithm is possible that, given a program p that always halts, checks whether this program always compute n!

$$\forall n(p(n)=n!)$$

Proof. By contradiction. Let's assume that factorial-checker exists.

$$\text{fact-checker}(p)=\begin{cases}1 & \text{if } \forall n(p(n)=n!)\\ \emptyset & \text{if } \exists n(p(n)\neq n!)\\ \text{whatever} & \text{if p does not always halt.}\end{cases}$$

Build a zero-checker based on this program

$$g \xrightarrow{?} \forall n(g(n)=\emptyset)$$

$$p(n)=g(n)+n! \xrightarrow[\text{fact checker}]{} \forall n(p(n)=n!)$$

$$\text{zero-checker}(g)=\text{fact-checker}(g(n)+n!)$$

```
public static int p(int n)
{
    return g(n) + fact(n);
}
```

$g(n) = \emptyset \iff g(n) + n! = n!$

if $g(n) = \emptyset$ then $p(n) = g(n) + n! = n!$

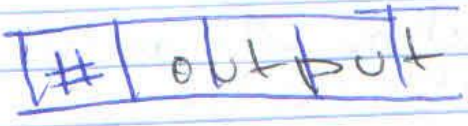
if $\exists n (g(n) \neq \emptyset)$ then $p(n) = g(n) + n! \neq n!$

Turing machine for computing

~~## input~~

non-negative numbers:
binary 110
unary numbers
1 11 111 1111

# denotes empty

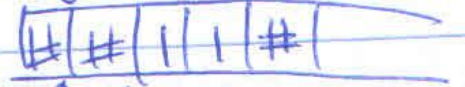


number 2

(2,3)

(2,0)

(0,2)



(0,0)

adding 1 in unary code $n \geq 2$

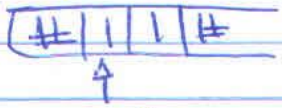
 we have

 we want

(start, #) → (R, working)

(working, 1) → R
(working, #) → (L, 1, back)
(back, 1) → L
(back, #) → halt


start


working


working

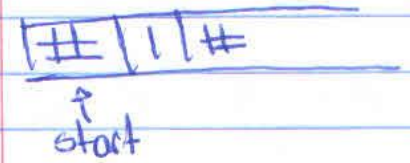

working


back

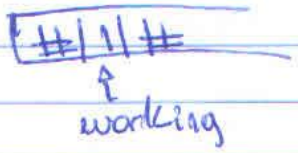

back


~~back~~ halt

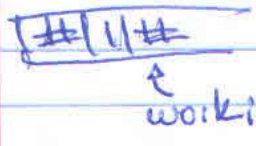
n=1



(start, #) → (R, working)

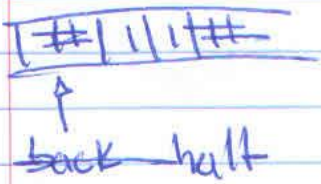


(working, 1) → R



(working, #) → (L, 1, back)

(back, 1) → L



(back, #) → halt

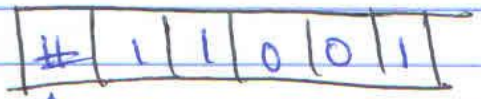
Representing # in TM's

10011

Algorithm for adding 1 in binary code:
Start with smallest bit
If you see 1 you replace it with 0 and continue
Until you see 0 or ␣ and replace with 1

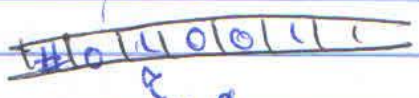
$$\begin{array}{r} 11 \\ +\ 1 \\ \hline 100 \end{array} \qquad \begin{array}{r} 101 \\ +\ 1 \\ \hline 110 \end{array} \qquad \begin{array}{r} 110 \\ +\ 1 \\ \hline 111 \end{array}$$

In a TM



start

r1w0



r1w0

r1w0

back

back

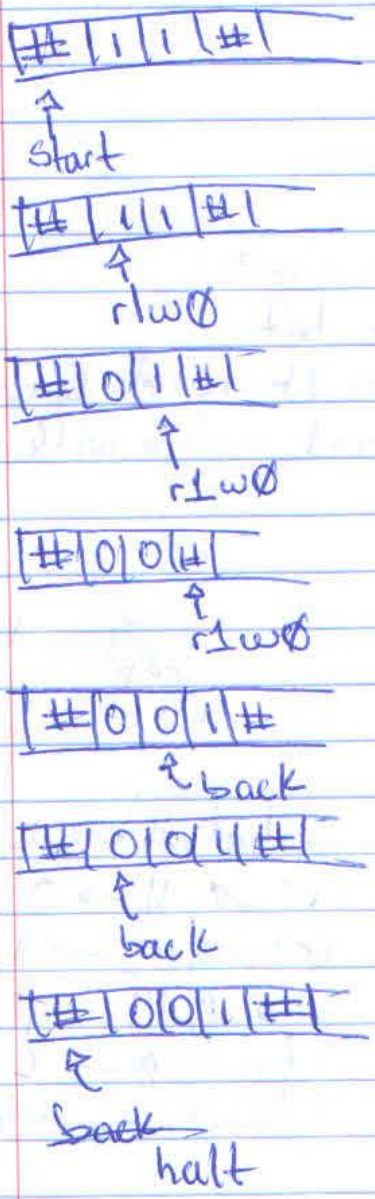
replace 1 with 0

(start, #) → R, r1w0
(r1w0, 1) → 0, R
(r1w0, 0) → 1, L, back
(r1w0, #) → 1, L, back
(back, 0) → L
(back, #) → halt

back

halt

n=3 in a TM



adding unary numbers.

we have

we want

(start, #) → ln 1st number, R

in1num, 1 → R

in1num, # → 1, R, in2num

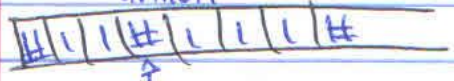
in2num, 1 → R

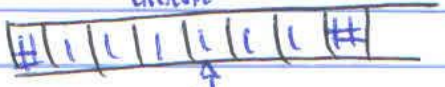
in2num, # → 1, ready

ready, 1 → #, L, rewind

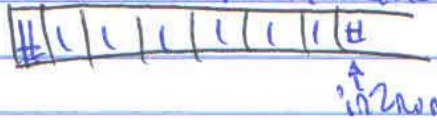
rewind, 1 → L

rewind, # → halt





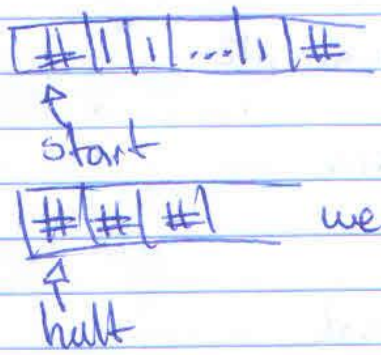
(and so on)



(and so on)
until halt

Th. A function is computable by a TM if and only if it is μ-recursive
(≡ Java program)

How to represent $\emptyset$.



start, # → working, R
working, 1 → R
working, # → erasing, L
erasing, 1 → #, L
erasing, # → Halt