

Theory of computation

Week 2

$$m, n \in \mathbb{N}$$

$$\text{mult} = 0$$

for ($i = 0; i < m, i++$) {

$\text{mult} = \text{mult} + n$ }

$$\text{mul} + (m, m)$$

$$\left\{ \begin{array}{l} \text{mul} + (m, 0) = 0 \\ \text{mul} + (m, n+1) = \text{mul} + (m, n) + n \end{array} \right.$$

$$\left\{ \begin{array}{l} f(n_1 \dots n_k, 0) = g(n_1 \dots n_k) \\ f(n_1 \dots n_k, m+1) = h(n_1 \dots n_k, m, f(n_1 \dots n_k, m)) \end{array} \right.$$

$$k=1$$

$$f(m, 0) = g(m)$$

$$f(m, m+1) = h(m, m, f(m, m))$$

$$\left\{ \begin{array}{l} f(n_1, 0) = 0 \\ f(n_1, m+1) = f(n_1, m) + n_1 \end{array} \right.$$

$$g(n_1) = 0$$

$$h(n_1, m, f) = f + n_1 = \pi_3^3 + \pi_1^3$$

$$= \text{add}(\pi_3^3, \pi_1^3)$$

$f = \text{primitive recursion } (g, h)$

$$\text{mult} = \text{PR}(0, \text{add}(\pi_3^3, \pi_1^3))$$

⊕ Square

$$0^2 = 0$$

$$(m+1)^2 = m^2 + 2m + 1$$

$$\text{square}(0) = 0$$

$$\text{square}(m+1) = \text{square}(m) + 2m + 1$$

$$f(0) = 0 \quad g = 0$$

$$f(m+1) = f(m) + m + m + 1$$

$$h =$$

Prove factorial is descriptive ~~and~~ recursive.

$$\text{fact}(n) = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

```

fact = 1;
factorial(n) {
  return factorial(n-1);
  for (i=1; i <= n; i++) {
    fact = fact * i;
  }

```

$$f(0) = 1;$$

$$f(m+1) = f(m) \times (m+1)$$

1. Write java code \rightarrow loop // General Description
2. -
3. Match with definition

~~$f(m_1, \dots, m_k)$~~


~~$f(m_1, \dots, m_k)$~~

$$f(m_1, \dots, m_k, 0) = g(m_1, \dots, m_k)$$

$$f(m_1, \dots, m_k, m_{k+1}) = g(m_1, \dots, m_k, m_{k+1})$$

$$g(m_1, \dots, m_k, m_{k+1})$$

$$f(m_1, \dots, m_k, m_{k+1})$$

 Before Thursday

- 1) a^n is PR
- 2) $n^3 + n^2 + 5$ is PR
- 3) $n^3 - n^2 + 5$ is PR
- 4) Exclusive OR is PR

Primitive Recursion

⊕ Prove that

$$a - b = \begin{cases} a - b & \text{if } a > b \\ 0 & \text{if } a \leq b \end{cases}$$

$$\text{prev}(3) = 2$$

$$\text{prev}(2) = 1$$

$$\text{prev}(1) = 0$$

$$\text{prev}(0) = 0$$

$$\begin{cases} f(0) = g \\ f(m+1) = h(m, f(m)) \end{cases}$$

$$\begin{cases} \text{prev}(0) = 0 \\ \text{prev}(m+1) = 1 \end{cases}$$

$$\begin{aligned} g &= 0 \\ h(m, f) &= m \\ h &= \pi_1^2 \end{aligned}$$

$$\Rightarrow \text{prev} = \text{PR}(0, \pi_1^2)$$

$$\text{PR}(0, \pi_1^2) \rightarrow k = 0$$

$$a - b = \underbrace{(((a - 1) - 1) - 1)}_{b \text{ times}}$$

$$f \text{ minus} = a$$

for $(i = 1; i < b; i++)$ {

$$\text{minus} = \text{minus} - 1;$$

}

$$\begin{cases} f \text{ minus}(0) = a \\ f \text{ minus}(a, m+1) = h(f(a, m)) \end{cases}$$

$$f(n_1, 0) = n_1$$

$$f(n_1, m+1) = \text{prev}(f(n_1, m)) \\ h(n, m, f)$$



Prove

$$g = n = \pi^3_1$$

Conditions — if then

$<, >, =, <=, >=$

+ > 0 and, not

true $\equiv 1$

false $\equiv 0$

$\begin{cases} \text{pos}(0) = 0 - g \\ \text{pos}(m-u) = 1 - h \end{cases}$

$$\Rightarrow \text{PR}(g, h) = \text{PR}(0, 1)$$

> 0 is PR

+ $a > b$

$$\Rightarrow a - b > 0$$

$$a - b \equiv \text{PR}(\text{pos}(a - b))$$

$\begin{cases} \text{not}(0) = 1 \\ \text{not}(m-u) = 0 \end{cases}$

$a \leq b$ $\stackrel{\text{not}}{\equiv} \text{pos}(a > b)$
not is PR.

$\left. \begin{array}{l} n > 0 \text{ is PR} \\ a > b \equiv \text{pos}(a - b) \\ \text{not is PR} \\ a \geq b \equiv \text{not}(b > a) \end{array} \right\}$

$$a = b \Leftrightarrow \text{not}(a > b) \ \&\& \ \text{not}(a < b)$$

$$a \vee b \Leftrightarrow \text{not}(\text{not}(a), \text{not}(b))$$

- Three classes of basic functions:
 - 2 ways of building new PR f
 - successor
 - zero
 - projections

composition
PR.

Projection function: $p_i^k : \mathbb{N}^k \rightarrow \mathbb{N}$.

$$p(x_1, x_2, \dots, x_k) = x_i$$

↓ it goes from k -dimensional "space" into 1-dimensional
→ basic building blocks & PR f.

If g_1, g_2, \dots, g_m are functions $\mathbb{N}^k \rightarrow \mathbb{N}$, and f is a function $\mathbb{N}^m \rightarrow \mathbb{N}$
 $\Rightarrow h : \mathbb{N}^k \rightarrow \mathbb{N}$ given by:

$$h(x_1, x_2, \dots, x_k) = f(g_1(x_1, \dots, x_k), \dots, g_m(x_1, \dots, x_k))$$

$\Rightarrow h$ arises from:

$$h = C_n[f, g_1, \dots, g_m]$$

$$h(x_1, x_2, x_3) = f(g(x_1, x_2, x_3))$$

Primitive recursion:

$$f : \mathbb{N}^k \rightarrow \mathbb{N}$$

$$g : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$$

then $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ is said to be defined by PR from

f and g if:

$$h(\bar{x}, 0) = f(\bar{x})$$

$$h(\bar{x}, s(y)) = g(\bar{x}, y, h(\bar{x}, y))$$

$$h = PR(f, g)$$

* Def:

- Any PR function: can be obtained from 0, S, π by composition and PR
- 0 is PR, S, π are so.

A polynomial is a f. can be obtained from const or variables by using multiplication / addition.
 (*) (+)

Ex: $n^3 + n^2 + 5 \Rightarrow n^2$: result of mult $\times \rightarrow$ PR
 $n^3 + n^2 + 5$: result of + \rightarrow PR

Ans: Exclusive OR,

A	B	A XOR B	+ Two ways:
0	0	0	(1) $A \text{ XOR } B \equiv (A \vee B) \wedge \neg (A \wedge B)$
0	1	1	
1	0	1	and $(\text{or}(A, B), \text{not}(\text{and}(A, B)))$
1	1	0	(2) $A\bar{B} + \bar{A}B$ $(\neg A \wedge B) \vee (A \wedge \neg B)$

Logical notations:

<, \leq , >, \geq , =, \neq .

and, or, not

- if then statement

$(a > 0) ? a : -a$

\Leftrightarrow if $(a > 0)$ then a else -a;

$h(n) = \text{if } p(n) \text{ then } s(n) \text{ else } g(n)$

$\begin{array}{ccc} | & | & | \\ \text{PR} & \text{PR} & \text{PR} \end{array}$

We want to prove $h(n)$ is PR.

let's make it natural.

Either $p(n)$ and $f(n)$ is the value.
or $\neg p(n)$ and $g(n)$ is the value.

$$\text{not } P = 1 - P$$

$$A \& B \equiv A * B$$

$$A \vee B \equiv \neg A * B + A + B$$

$$p(n) * f(n) + \underbrace{(1 - p(n))}_{\text{not}} * g(n)$$

if $p(n)$ is true $\Rightarrow p(n) = 1 : 1 * f(n) + 0 * g(n) = f(n)$
false $\Rightarrow p(n) = 0 : 0 * f(n) + 1 * g(n) = g(n)$



By Tuesday

1) Prove that this is PR.

if $\bar{p}(n)$ then $\bar{f}(n)$

else if $\bar{q}(n)$ then $\bar{g}(n)$

else $h(n)$

(+) Remainder:

$$\text{rem}(n) = 0$$

$$(m+1) \text{ rem } n = \begin{cases} (m \text{ rem } n + 1) & \text{if } (m \text{ rem } n + 1) < n \\ 0 & \text{otherwise} \end{cases}$$

$$0 \% 5 = 0 \rightarrow \text{prev. remainders}$$

$$1 \% 5 = 0 + 1$$

$$2 \quad 1 + 1$$

$$3 \quad 2 + 1$$

$$\text{rem}(n, m+1) = \begin{cases} \text{if } (\text{rem}(n, m) + 1 < n) & \text{then} \\ \text{rem}(n, m) + 1 & \text{else } 0; \end{cases}$$

$$f(n, m+1) = h(m, n, f(n, m))$$

||

$$\begin{cases} \text{if } (f+1 < n) & \text{then } f+1 \\ & \text{else } 0 \end{cases}$$

sof

qof

⊕ + Division.

$$0/n = 0$$

$$m+1/n = \begin{cases} \text{if } (m+1) \% n = 0 & \text{then } (m+1)/n \\ \text{else } m/n \end{cases}$$

$$13/3 = 4$$

$$15/3 = 5$$

$$17/3 = 5$$

$$14/3 = 4$$

$$16/3 = 5$$

$$18/3 = 6$$

$$f(n, 0) = 0$$

$$f(n, m+1) = \begin{cases} \text{if } (m+1) \% n = 0 & \text{then } f(n, m) + 1 \\ \text{else } f(n, m) \end{cases}$$

$$h(n, m, f(n, m))$$

⊕ Is every computable function : p recursive ?

2 proofs :

< more detailed, easier proof

less detailed, more natural



2)

Theorem : there exists a computable function which is not PR.

Way 1: PR $f(n) \dots$ PR $(0, \sigma, \pi)$
LaTeX PR $(0, \backslash errormessage, \backslash error, \backslash pi-1)$

↓
 $0 + \dots + 1$

↓
 $C \equiv$ code of a PR $f(n)$

+ Computable but not PR

$f(n) = \begin{cases} \text{if } n \text{ is a valid code} \\ \text{then } f(n) + 1 \\ \text{else } 0 \end{cases}$

Statement: it is not PR
proof by contradiction

let f be PR
then it has a code c .

$$f(n) = f_c(n) \Rightarrow f(c) = f_c(c)$$

by def since c is a valid code.
then apply $f_c(c) = f_c(c) + 1$

$$\Rightarrow f(c) = f(c) + 1$$

$$\Rightarrow 0 = 1$$

contra.



Reproduce the proof :

→ Theorem: not every computable function is PR.

we will construct a function $f(n)$

which is

→ computable

→ not PR.

+ 1st part: codes of P.R. f's (PR functions)

Goal: assign to every PR fn. a number.

We start with an expression PR (π_1, σ_0)

↓ use LaTeX translate formula to strings

PR ($\backslash pi 1$, $\backslash sigma 0$)

↓ ASCII

strings of binary 0, 1 : $0110 \dots 011$

↓ tuck put 1 in front

①0110 ... 1 ... 1 ... 1 ... 1

(in order to differentiate the code)

①	= 3		add 1s	1 = 1
①01	= 5		if not we'd have	1 = 1
①001	= 9			1 = 1

↓

... code of a PR fn.

→ Statement: There is an algorithm that given a natural number c .

* checks whether c is a valid code of a PR. fn.

* if it's produces a java code for for computing the corresponding p.r. fn.

* Idea :

1. Strip off the ① in front of every piece of code.
2. Check if all combinations are ASCII symbols
3. LaTeX compiler checks if the syntax is correct.
4. Check syntax.

$$+ h = PR(\hat{f}, \hat{g})$$

$$h = f_i$$

for ($i=0; i < m; i++$)

$$h = g_i$$

+ $f \circ g_i$

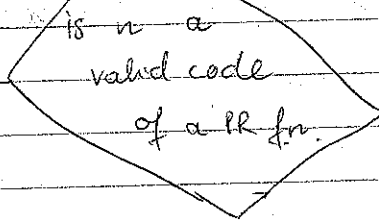
compute f_i

compute g_i

n

* n :

no



return 0

yes

apply f_n
to n

// This shows

f is computable

return

$f_n(n) + 1$

$$f(n) \stackrel{\text{def}}{=} \begin{cases} f_n(n) + 1 & \text{if } n \text{ is a valid code} \\ 0 & \text{otherwise} \end{cases} \text{ of a PR f.n.}$$

We assume that f is P.R.

Then f is represented by some exp., that exp. has a code c .

$$f_c(x) = f(x)$$

Since c is valid code, by definition of f , we have

$$f(c) = f_c(c) + 1$$

In particular for $x = c$

$$f(c) = f_c(c)$$

$$\Rightarrow f_c(c) + 1 = f_c(c)$$

$$\Rightarrow 1 = 0$$

(incorrect)

\rightarrow by contradiction we've shown that f is not P.R.

Georg Cantor

1845 - 1918

Cantor's proof: we only care about the existence
 pr. prove: we also care about computable.

diagonalization

Result: \exists a computable for which is not P.R.

- 1st proof easier to describe, but f is meaningless.

- 2nd proof, f is meaningful but the proof is more difficult.

⊕ Reminder:

We started with σ

$$a \cdot b = \underbrace{a + 1 + 1 + 1 + 1}_{b \text{ times}}$$

$$\underbrace{a + a + a + a \dots}_{b \text{ times}} = a \times b$$

$$\underbrace{a \times a \times a \times a \dots}_{b \text{ times}} = a^b$$

level 0: $f_0(n, m) = n + 1$

level 1: $f_0(n, f_0(n, \dots))$

$$f_1(n, 0) = 0$$

$$f_1(n, m+1) = f_0(f_1(n, m), \overset{n}{f_1(n, m)})$$

$$f_2(n, m+1) = f_1(f_2(n, m), n)$$

$$f_3(n, m+1) = f_2(f_3(n, m), n)$$

...

$$f_{k+1}(n, m+1) = f_k(f_{k+1}(n, m), n)$$

→ Archimedes.

$$a \overbrace{a \overbrace{a \dots}^b}^{b \text{ times}} = a^b$$

$$A(n) = f_n(n, n)$$

Ackermann's function.

Ackermann

$$A(0) = f_0(0, 0) = 1$$

$$A(1) = f_1(1, 1) = 1 + 1 = 2$$

$$A(2) = f_2(2, 2) = 2 \times 2 = 4$$

$$A(3) = f_3(3, 3) = 3^3 = 27$$

$$A(4) = f_4(4, 4) = 4^{4^4} = 4^{(10^3)} \approx 10^3$$

- Wed: 6-7. R. 300.

▶ - By Tuesday: Ask Dr. VK → theory project topic

Prove that $A(n)$ is not PR.

$$A(n) = f_n(n, n)$$

if $\leq k$ for-loops $\Rightarrow f \leq A_k(n, n)$

→ Don't use ~~for-loops~~ to prove.

$$0, \sigma(0), \sigma(\sigma(0))$$

$$\tau_1^1, \sigma(\tau_1^1), \sigma(\sigma(\tau_1^1))$$

$$n, n+1, n+2$$

∃ cst

$$f(n) \leq n + c$$

$$f(n_1, n_2, \dots, n_k) \leq n_1 + c$$

$$f(\bar{n}, 0) = g(\bar{n})$$

$$f(\bar{n}, m+1) = h(\bar{n}, m, f(\bar{n}, m))$$

$$f(0) = g$$

$$f(m+1) = h(m, f(m)) \leq \max(m, f(m)) + c$$

|| c, constant

$$f(0) = g$$

$$f(0+1) = h(1, f(0)) + c \leq \max(1, g) + c \leq \tilde{g} + c$$

$$f(1+1) = h(2, f(1)) + c \leq \max(2, \tilde{g} + c) \leq \tilde{g} + 2c$$

for no for-loops

$$f(n) \leq \underbrace{n+1+\dots+1}_{\text{const times}}$$

for one for-loop:

$$f(n) \leq \underbrace{n+\dots+n}_{\text{const times}} + \tilde{g}$$

const times

$$f_0: n+1$$

$$f_1: n+2n$$

$$f_2: n^2$$

$$\dots$$

$$f: n^n$$



1. Turn in: project ~~the~~ proposal

A-PR

2. Prove that not every computable fn is a P.R

$A(n)$

▽

First hypothesis: Every computable fn. is PR.

Fact: There exists a computable fn is NOT PR.

(Ex)

: Ackermann fn $A(n)$.

Natural idea:

Add $A(n)$ to basic fn, 0 , S , π^k , and $\lambda A(n)$.

if-then ✓ missing while-loop
 for ✓ we need to formalize while-loop
 boolean ✓ Big diff: # of iterations is not
 ... explicitly known. determined by
 ... a condition

while (!P)
 { ... }

of iterations \equiv smallest μ for which $f(\bar{n}, m)$ is true.
 μ -recursion:

$$f(n_1, \dots, n_k) = \mu m P(n_1, n_2, \dots, n_k, m)$$

\downarrow
smallest

μ = "myu":

→ A μ -recursive fn is any fn which can be obtained from 0, σ , π_k^k , by σ , PR and μ -recursion.

$$f(\bar{n}, m) = \mu m. P(\bar{n}, m)$$

$$a - b = c \text{ s.t. (such that)}$$

$$c + b = a$$

$$\mu c (c + b = a)$$

"smallest $c = \mu c$ "

$m = 0$
 while (!P)
 {
 m++;
 }

$$\mu c (c + b \geq a) : a \leq b : \mu c = 0$$

$$\mu c (c + b \geq a) : a > b$$

123

3. describe division as μ -recursive fn.
 a/b .

4. describe a fn.. $f(n) = \begin{cases} 1 & \text{if } n = 2 \\ 2 & \text{if } n = 1 \\ \text{undef.} & \text{otherwise} \end{cases}$

$f(n) =$

$\mu_m (0 = 1)$ $m = 0$
while $(!(0 = 1))$
{ $m++$; }

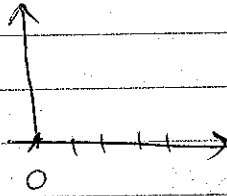
This fn. is never defined.

$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$

$\mu_m (n = 0 \ \& \ m = 0)$

if $n = 0$ then return 0

if $n \neq 0$ then undef.



$f(n) = \begin{cases} 1 & \text{if } n = 2 \\ 2 & \text{if } n = 3 \\ \text{undef} & \text{otherwise} \end{cases}$

$\mu_n [(n = 2) \ \& \ (m = 1) \ \vee$
 $(n = 3) \ \& \ (m = 2)]$

m : smallest number
makes fn. satisfied

→ μ -recursive functions may be undefined.

Totally recursive \equiv

μ -recursive and everywhere defined

Every Java program can be described as a μ -recursive fn.

! Important

In 1930s, 2 different defs appeared

- recursion (church) based on a prog. lang.

- Turing: low-level: based on step-by-step operation
of a computing device.

Reproduce = write detailed proof.

$$f_n(n) + 1$$

+ proof has to be completely clear.

Computable \equiv μ -recursive.

Big problems with while-loops: \leftarrow go into infinite loops
may not stop

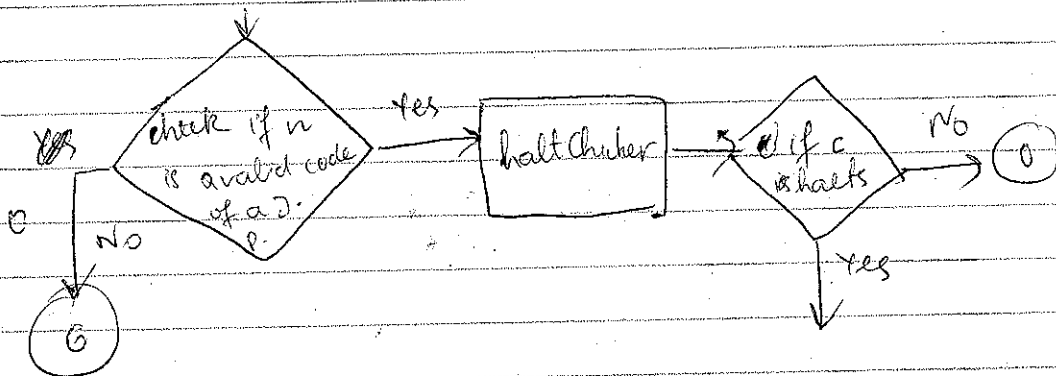
// halt, queue \leftrightarrow stop, stand in line
UK US

Theorem: NO algorithm is possible, that given a program p and data d , would check whether p halts on d (data) or not.

haltechecker(p, d) = $\begin{cases} \text{yes} & \text{if } p \text{ halts on } d. \\ \text{no} & \text{otherwise.} \end{cases}$

Reduction to a contradiction. let's assume that haltechecker exists.

$f_n(n) = \begin{cases} f_n(n) + 1 & \text{if } n \text{ is a valid Java program} \\ 0 & \text{otherwise} \end{cases}$



- With $x = c \rightarrow f_c(x) = f(x) \rightarrow f_c(c) = f(c)$ (2)

- By definition of f , since c is a valid code, f_c always halts \rightarrow

$$f_c(c) = f(c) + 1 \quad (1)$$

(1), (2) \rightarrow contradiction.

- Assume: if f is computable then there is a code c , by some Java program, let's denote by (c) , the code of Java program, particular $x = c$

$$f_c(x) = f(x)$$



1) Redo proof.

2) Reproduce halting proof.

A program

1) does it halt?

2) if it halts, does it produce correct results?

we want $\forall x (f(x) = 0)$.

```
public static int zero(int x) {  
    return 0;  
}
```

Def. A zero checker is a prog that given p that always halts, checks whether $\forall x p(x) = 0$.

zero checker(p) = $\begin{cases} \text{yes} & \text{if } \forall x p(x) = 0 \\ \text{no} & \text{otherwise} \end{cases}$

Theorem: Zero checkers do not exist.

1. Assume zero checkers exist.
2. Build halt checkers on top zero checkers \rightarrow halt checkers exist!
3. However, halt checkers don't exist \rightarrow neither do zero checkers.

We assume that zero checkers exist. and we will conclude that a halt checker exists.

$$\text{halt checker}(p, d) = \begin{cases} 1 & \text{if } p \text{ halts on } d \\ 0 & \text{if } p \text{ does NOT halt on } d. \end{cases}$$

we can design:

$$\text{halt}(p, d, t) = \begin{cases} 1 & \text{if } p \text{ halts on } d \text{ by time } (t) \\ 0 & \text{otherwise.} \end{cases}$$

p halts on $d \equiv$ there exists a t that p halts on d
 p does NOT $\equiv \forall t$ p does NOT halt on d .

we define:

$$f_{p,d}(t) = \text{halt}(p, d, t)$$

Zero checker: $f_{p,d} = 0$

- Apply zero checker to $f_{p,d}$

$$\text{halt checker} = \neg \text{Zero checker}(f_{p,d})$$



3. Reproduce zero checker proof.

$$\text{Square checker}(p) = \begin{cases} 1 & \text{if } \forall x \quad p(x) = x^2 \\ 0 & \text{otherwise} \end{cases}$$

Theorem: square checker is impossible.

Proof: we assume that square checker exists and we will build a zero checker.

$$q \xrightarrow{\text{square checker}} \textcircled{1} \quad \forall x \quad q(x) = x^2$$

$$\text{zero checker}(p) \stackrel{\text{def}}{=} \text{square checker}(p + x^2)$$

$$\forall x \quad p(x) + x^2 = x^2$$

$$\Downarrow$$

$$\forall x \quad (p(x)) = 0$$

4. - prove a μ -checker ^{does} not exist.

μ -recursion.

+ Open problem:

$$n \begin{cases} \text{if } n \text{ is even} & n = n/2 \\ \text{if } n \text{ is odd} & n = 3n + 1 \end{cases}$$

$$a/b = \mu c (c(b+1) > a)$$

⊕ church μ -recursive

$0, \sigma, \pi^k$

$0, RR, \mu$ -recursive

↓
for-loop

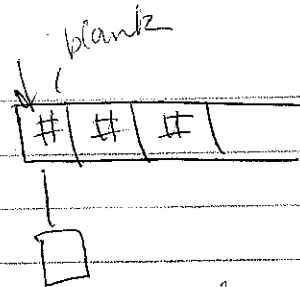
↓
while-loop

Turing Machines:

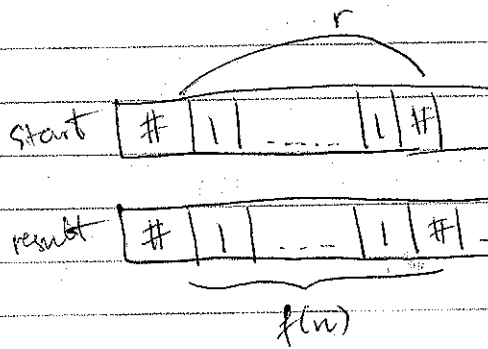
can we compute every μ -recursive fn on a Turing machine?

Unary code :

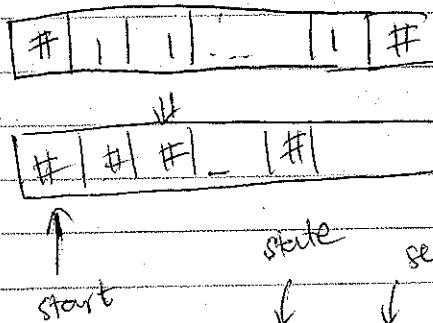
0 1 1
 2 11
 3 111



not computable on Turing



machine if there exists on Turing machine with the following property:



⊕ Function: return 0;

- (start, #) → (R, reading)
- (reading, 1) → (R, reading)
- (reading, #) → (L, erasing)
- (erasing, 1) → (L, #, erasing) — replace 1 by #
- (erasing, #) → halt

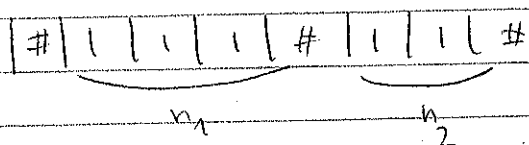
⊕ Function: Add 1: ♂

- (start, #) → (R, reading)
- (reading, 1) → (R, reading)
- (reading, #) → (L, 1, back)
- (back, 1) → (L, back)
- (back, #) → halt



1. Design a TM that computes $f(n) = n+2$.
 // Use the 'state' wisely.

+ function projection $\Pi_1^2 = (n_1, n_2) \rightarrow n_1$



(start, #) \rightarrow (R, in1right)

(in1right, 1) \rightarrow (R, in1right)

(in1right, #) \rightarrow (R, in2right)

(in2right, 1) \rightarrow (R, in2right)

(in2right, #) \rightarrow (L, #, erase2no)
 \downarrow
 replace

(erase2no, 1) \rightarrow (L, erase2no)

(erase2no, #) \rightarrow (L, in1left)

(in1left, 1) \rightarrow (L, in1left)

(in1left, #) \rightarrow halt

2. TM: Π_1^3 $f(n_1, n_2, n_3) = n_1$

3. Extra credit: Π_2^2 $f(n_1, n_2) = n_2$