

$$f(n_1, n_2) = n_2 \pi_2^2$$

(start, #) \rightarrow (R, 1, erasing1)
 (erasing1, 1) \rightarrow (R, #, erasing1)
 (erasing1, #) \rightarrow (R, forward2)
 (forward2, 1) \rightarrow (~~forward2~~, (R, forward2))
 (forward2, #) \rightarrow (L, backtrack)
 (backtrack, 1) \rightarrow (L, #, carry1)
 (backtrack, #) \rightarrow (
 (carry1, 1) \rightarrow (L, carry1)
 (carry1, #) \rightarrow (E, check next blank)
 (check next blank, 1) \rightarrow (R, #, final clearing)
 (final clearing, #) \rightarrow (final move, 1, L)
 (final move, #) \rightarrow halt

(check next blank, #) \rightarrow (go right, R)
 (go right, #) \rightarrow (R, 1, forward2)



π_2^3 : Define a TM for $(n_1, n_2, n_3) = n_2^2$

square checker $\Leftrightarrow \forall x (q(x) = x^2)$

zero checker $\Leftrightarrow \forall x (p(x) = 0)$

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

$$\text{Since } \forall x, p(x) + x^2 = x^2$$

$$\Rightarrow p(x) = 0$$

9, 2009

why do we have to rewind? // computation

- computable \rightarrow has an algorithm

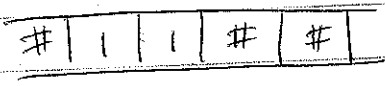
- Recursive: $\langle 0, 1, \pi^k \rangle$
 $\langle 0, \mathbb{P}\mathbb{R} \rangle$, recursion

(F) TM \rightarrow combination (C)

$n \rightarrow n+1$ \downarrow (f)



$n \rightarrow n+2$ \downarrow (g)



- (start_f, #) \rightarrow (right_f, R)
- (right_f, 1) \rightarrow (right_f, R)
- (right_f, #) \rightarrow (back_f, 1, L)
- (back_f, 1) \rightarrow (back_f, L)
- (back_f, #) \rightarrow halt.

- (start_g, #) \rightarrow (right_g, R)
- (right_g, 1) \rightarrow (right_g, R)
- (right_g, #) \rightarrow (putting 1_g, 1, R)
- (putting 1_g, #) \rightarrow (back_g, 1, L)
- (back_g, 1) \rightarrow (back_g, L)
- (back_g, #) \rightarrow halt.

$h(n) = g(f(n))$ composition.

+ Idea: first run TM for f and then run TM for g

We can't combine the rules of f and g

- 1) we have different conclusions for the same
- 2) same names can be used on both TM with slightly different things.
- 3) we can't have "halts" after the 1st TM.

+ Solution.

1) rename states: $\left. \begin{array}{l} \text{start} \rightarrow \text{start}_f \text{ (1st TM)} \\ \text{start} \rightarrow \text{start}_g \text{ (2nd)} \end{array} \right\}$

2. change halt_f \rightarrow start_g
 \Rightarrow This is a combination of f & g.

3) Take any two TMs and make a combination

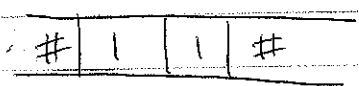
(+) TM \rightarrow Primitive Recursion (PR)

$h = PR(f, g)$

$h = f$
 for (int i=0; i < n; i++)
 $h = g(x, f(i))$

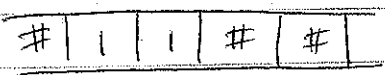
(+) TM \rightarrow combination (c)

$n \rightarrow n+1$ \swarrow (f)



- $(start_f, \#) \rightarrow (right_f, R)$
- $(right_f, 1) \rightarrow (right_f, R)$
- $(right_f, \#) \rightarrow (back_f, 1, L)$
- $(back_f, 1) \rightarrow (back_f, L)$
- $(back_f, \#) \rightarrow \& \text{halt}$

$n \rightarrow n+2$ \swarrow (g)



- $(start_g, \#) \rightarrow (right_g, R)$
- $(right_g, 1) \rightarrow (right_g, R)$
- $(right_g, \#) \rightarrow (putting_1_g, 1, R)$
- $(putting_1_g, \#) \rightarrow (back_g, 1, L)$
- $(back_g, 1) \rightarrow (back_g, L)$
- $(back_g, \#) \rightarrow \text{halt}$

$h(n) = g(f(n))$ composition.

+ Idea: first run TM for f and then run TM for g

We can't combine the rules of f and g

- 1) we have different conclusions for the same
- 2) Same names can be used on both TM with slightly different things
- 3) we can't have "halts" after the 1st TM.

+ Solution.

- 1) Rename states: $\left. \begin{array}{l} \text{start} \rightarrow \text{start}_f \text{ (1st TM)} \\ \text{start} \rightarrow \text{start}_g \text{ (2nd)} \end{array} \right\}$
2. change $halt_f \rightarrow start_g$
 \Rightarrow This is a combination of g.



1) Take any two TMs and make a combination

(+) TM \rightarrow Primitive Recursion (PR)

$h = PR(f, g)$

```

h = f
for (int i=0; i < n; i++)
{
    h = g(x, f(i))
}
    
```

$$h(a^{\rightarrow}, 0) = f(a^{\rightarrow})$$

$$h(a^{\rightarrow}, n+1) = g(a^{\rightarrow}, h(a^{\rightarrow}, n))$$

| a^{\rightarrow} | # | n | a^{\rightarrow} | $n-1$ | # | a^{\rightarrow} | # | $n-2$ | ... | # | a^{\rightarrow} | # | 0

| a^{\rightarrow} | # | | | | # | a^{\rightarrow} | # | | | # | a^{\rightarrow}

⊕ TM \rightarrow μ -recursion
 $\mu m.p(a^{\rightarrow}, m)$: P: property returns $\begin{matrix} 1 \\ \leftarrow \\ 0 \end{matrix}$
 \downarrow
 apply while loop \rightarrow TM.

So far, we have two languages:


- Recursive fn
- TM: bitwise

There is another intuition: sets

$$N = \{0, 1, 2, 3, \dots\}$$

$$A \subseteq N$$

Def: A set 'A' is called "decidable" if there is an algorithm that given a natural # decides whether $n \in A$.

- 1) \emptyset algorithm returns no
- 2) N always returns yes.
- 3) a finite set $\{n_1, n_2, n_k\}$.
- 4) if $\begin{matrix} A \\ \swarrow \\ B \end{matrix}$ is decidable $\rightarrow A \cup B$ is decidable?
- 5) $\cap \rightarrow$ 
 - check if $n \in A \xrightarrow{\text{yes}}$ return "yes"
 - if not, check if $n \in B$

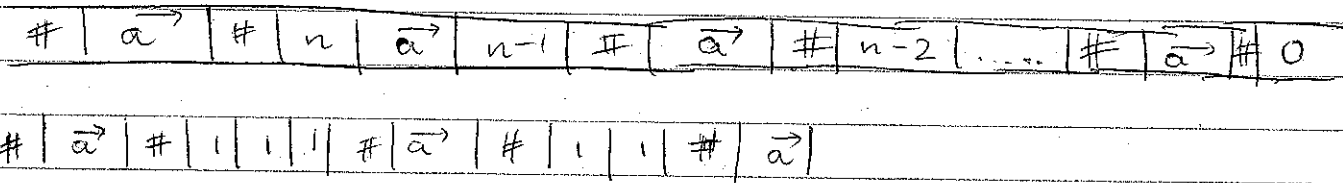


2. Prove if $A \cap B$ is decidable if $\begin{matrix} A \\ \swarrow \\ B \end{matrix}$ is decidable

6) complement: if A is decidable.
 is A^c is decidable?

$$h(\vec{a}, 0) = f(\vec{a})$$

$$h(\vec{a}, n+1) = g(\vec{a}, h(\vec{a}, n))$$



⊕ TM \rightarrow μ -recursion
 $\mu n, P(\vec{a}, n)$: P: property returns $\begin{matrix} 1 \\ < \\ 0 \end{matrix}$
 \downarrow
 apply while loop \rightarrow TM.

so far, we have two languages:

- Recursive fn
- TM: bitwise

There is another intuition: sets

$$N = \{0, 1, 2, 3, \dots\}$$

$$A \subseteq N$$

Def: A set 'A' is called "decidable" if there is an algorithm that given a natural # decides whether $a \in A$.

- 1) \emptyset algorithm returns no
- 2) N : always returns yes.
- 3) a finite set $\{n_1, n_2, n_k\}$.
- 4) if $\begin{matrix} A \\ \cup \\ B \end{matrix}$ is decidable $\Rightarrow A \cup B$ is decidable?
- 5) $\cap \rightarrow$
 - check if $n \in A$ $\xrightarrow{\text{yes}}$ return "yes".
 - if not, check if $n \in B$.



2. Prove if $A \cap B$ is decidable if $\begin{matrix} A \\ \cup \\ B \end{matrix}$ is decidable

6) complement: if A is decidable.

is A^c is decidable?

7) Is there any set that is not decidable?

$\{ \langle p, d \rangle : p \text{ halts on } d \} \rightarrow \text{not decidable.}$

$\{ p : p \text{ always returns } 0 \} \rightarrow \text{not decidable.}$

Def: A set 'A' is called recursively enumerable if there is an algorithm that eventually prints all elements of A.

$n = 0;$

while (true) {

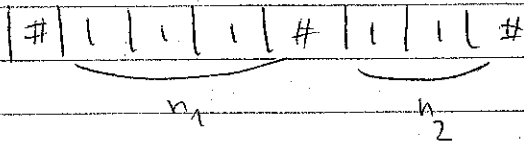
 System.out.println(n);⁶

}



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

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



- $(start, \#) \rightarrow (R, in1right)$
 $(in1right, \#) \rightarrow (R, in1right)$
 $(in1right, 1) \rightarrow (R, in2right)$
 $(in2right, 1) \rightarrow (R, in2right)$
 $(in2right, \#) \rightarrow (L, \#, erase2no)$
↓
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 $f(n_1, n_2) = n_2$

⊕ Fe

Feb 17, 2009

What is the code of a Java program?

- Java program - is a sequence of ASCII symbols.
- In machine code, every ASCII symbol is a sequence of 0s and 1s.
- Put 1 in front
- Get a natural #, this # is called the code of a Java program.

What f_c ?

Let's start with a natural # c . We step off 1s in front.

$f_c \rightarrow$ Java program
corresponding to # c .

$$f(n) = \begin{cases} f_n(n) + 1 & \text{if } n \text{ is a valid code and } f_n \\ & \text{halts on } n. \\ 0 & \text{otherwise} \end{cases}$$

check if: 1

Java program...

+ Assumption: A zero checker exists.

- Proof: Build a ~~zero~~ ^{halt} checker based on a zero checker.

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

$$+ !\text{haltChecker}(p,d) = \begin{cases} 1 & \text{if } f_{p,d}(t) = 0 \\ 0 & \text{otherwise} \end{cases}$$

→ We ~~do~~ need to use negation of halt checker since we can't compute when a program does halt? Hence, we have to compute when a program ^{does not} ~~never~~ halts by some time t .

$$\Rightarrow \text{haltChecker}(p, d) = \begin{cases} !\text{zeroChecker}(f_{p,d}) \\ 0 \end{cases}$$

$$\downarrow !\text{haltChecker}(p, d) = \begin{cases} 1 \\ 0 \end{cases}$$