

Theory of Computation. Kreinovich

nearest 100 weeks.

20/01/2009

Project Three types of project

Ideal: related to your area of interests, helpful in your thesis / diss.

Possible: take a paper & present it.

Alternative: Do some research in theory of computing.

All other classes

- given a problem

- you are expected to write a code for solving this problem.

1st stage: any code.

2nd stage: efficient code

3rd stage: problem is not (SE) originally precisely formulated

↙ If original requirements (specs) cannot be satisfied, then we need to discuss with a customer which specs can be relaxed.

→ intuition, experience.

Imp. problems

- what can be computed & what cannot be?

- If it can be computed then how? & on what devices it can be computed? how much resources?

- If it cannot be computed, how can we relax the problem to make it computable? in principle

what can be computed?

Equivalent form: what can be computed on any comp. device in

Example: write in Java

principle?

Aqs: we have a precisely formulated problem, we want to check whether this problem can be solved by a Java program?

Problem: Java is difficult to describe, so it is difficult to prove that something cannot be computed by a Java program.

[since Java is huge]

what we need: Simplify the notion of Java-Computable functions.

Trajectory of the class: (NP hardness) toughest

what can be computed?

* standard computers, Java

* we can use other devices.

Alonzo church: Church's Thesis: Anything that can be computed on any computational device can be also computed by a Java Program. - 1936.

Alan Turing: Turing m/c. - 1934
Church-Turing thesis: → mistake was found in 1936. & published

R. Grandy 1970s: mech. devices - church's thesis is true.

as advised: if quantum is used or electromagnet?

church thesis is a statement about the physical world.

If church's thesis is correct - then to prove that something is not computable on any physical device, we need to prove that it is not computable by a Java program.

lets start simplifying *

we have * objects, * operations (+, -, ...)

* if-then statements

* loops (for, while, ...)

* other constructions

[for loop when we know how many repetitions will be done]

* objects in Java simplest: natural numbers 0, 1, 2, 3, ...

String, Integers, double, arrays, characters

original obj: Integers.

Integers in a comp, everything is a sequence of 0's

now simplify the notion of Java computable

$$a^0 = 1, a^{n+1} = a^n * a$$

power = 1 ;
for (i = 0 ; i < n ; i++) } cs description

$$\text{power} = \text{power} * a;$$

$$\text{power}(a, 0) = 1;$$

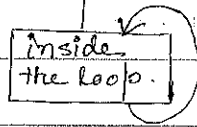
for (

$$\text{power}(a, i+1) = \text{power}(a, i) * a$$

expression

diagram

before loop → initialization



f = g ;
for (i = 0 ; i < m ; i++)

f = expression
↳ expression depending on f (previous value), on i, on some parameters

parameters don't change during computation, here "i" → parameters

$$f(n_1, \dots, n_k, m)$$

$$\left. \begin{aligned} f(n_1, \dots, n_k, 0) &= g(n_1, \dots, n_k) \\ f(n_1, \dots, n_k, m+1) &= h(n_1, \dots, n_k, f(n_1, \dots, n_k, m)) \end{aligned} \right\} \text{primitive recursion}$$

A. Church.

* power : k = 1
 $f(n_1, 0) = 1 \quad g(n_1) = 1$

$$f(n_1, m+1) = f(n_1, m) * n_1$$

$$h(n_1, m, f) = n_1 * f$$

$$\left. \begin{aligned} \text{fact}(0) &= 1 & f(0) &= 1 & g &= 1 & \text{here } k=0 \\ \text{fact}(i+1) &= \text{fact}(i) * (i+1) & f(m+1) &= (m+1) * f(m) \\ h(m, f) &= (m+1) * f \end{aligned} \right\}$$

Minor Notations:

++, --, +, -, *, /, %

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

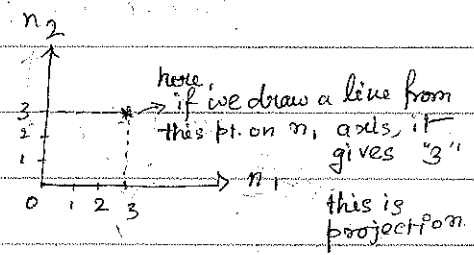
\sum sum

$$f(n) = n \quad \xrightarrow{n} \square \rightarrow n \quad \text{identity } f_0$$

Def. (version 0):

A. Primitive recursive (P.R) function is a function that can be obtained from 0, 1, 2, ...; identity f_0 , π_i^k , δ , +, -, *, /, %
 Id.

Projection $f(n_1, \dots, n_k) = n_i$ Projection $\equiv \pi_i^k$
 syntax $\rightarrow (\pi_i^k(n_1, \dots, n_k) = n_i \text{ by using composition \& PR})$



Qs: how can we further simplify this definition?

0 - probably stays

$$1 = \delta(0) \quad \delta(n) \equiv n + 1$$

2 = $\delta(\delta(0))$ conclusion: to describe which functions are p.r.

3 = $\delta(\delta(\delta(0)))$ δ (\equiv composition computable in a for loop);

it is sufficient to keep only one constant: (0).

caution: we are not talking about efficiency. size grows exponentially.

Unary notation for integers: $5_{10} = 1111$, $10_{10} = 11111111$

Binary notation: 1010_2

Id = π_1^1 (Identity is projecting a special case)

$$\pi_i^k(n_1, \dots, n_k) = n_i$$

Identity in terms of projection: $\pi_1^1(n_1) = n_1$

Addition: do we need to it as basic operation?

$$a + b = a + \underbrace{1 + \dots + 1}_{b \text{ times}} \quad \equiv \quad \text{sum} = a;$$

for $(i = 0; i < b; i++)$
 $\text{sum} += i;$

In terms of PR

$$\text{Sum}_{n_1, m}^m(a, 0) = a \quad \rightarrow \quad f(n_1, 0) = n_1 \quad g = \pi_1^1$$

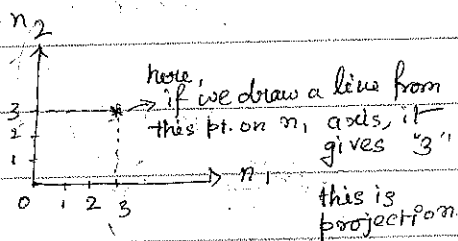
$$\text{Sum}_{n_1, m}^m(a, i+1) = \text{Sum}_{n_1, m}^m(a, i) + 1 \quad \rightarrow \quad f(n_1, m+1) = \delta(f(n_1, m)) \quad g(m) = m_1$$

$$f(n) = n \quad \xrightarrow{n} \square \rightarrow n \quad \text{identity } f_0$$

Def. (version 0):

A Primitive recursive (P.R) function is a function that can be obtained from 0, 1, ~~2, ...~~, identity f_0 , π_i^k , σ , $+$, $-$, $*$, $/$, $\%$ Id.

Projection $f(n_1, \dots, n_k) = n_i$ $\equiv \pi_i^k$
 syntax $\rightarrow (\pi_i^k(n_1, \dots, n_k) = n_i \text{ by using composition \& PR})$



Qs: how can we further simplify this definition?

0 - probably stays

$$1 = \sigma(0) \quad \sigma(n) = n + 1$$

2 = $\sigma(\sigma(0))$ Conclusion: to describe which functions are p.r.

3 = $\sigma(\sigma(\sigma(0)))$ σ (\equiv composition computable in a for loop);

it is sufficient to keep only one constant: (0).

Caution: we are not talking about efficiency. size grows exponentially.

Unary notation for integers: $5_{10} = ||||$, $10_{10} = |||||$

Binary notation: 1010_2

Id = π_1^1 (Identity is projecting a special case.)

$$\pi_i^k(n_1, \dots, n_k) = n_i$$

Identity in terms of projection: $\pi_1^1(n_1) = n_1$

Addition: do we need to it as basic operation?

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

$\sum_{i=0}^b a = a$
 for $(i=0, i < b, i++)$
 $\text{sum}++$

In terms of PR

$$\begin{aligned} \text{Sum}_{n_1, m}^m(a, 0) &= a & \rightarrow f(n_1, 0) &= n_1 \\ \text{Sum}_{n_1, m}^m(a, i+1) &= \text{Sum}_{n_1, m}^m(a, i) + 1 & \rightarrow f(n_1, m+1) &= \sigma(f(n_1, m)) \end{aligned}$$

$$g = \pi_1^1$$

$$g(m) = m$$

Def. (actual)

A P.R. function is any function that can be obtained from $0, \delta, \pi_i^k$ by using \circ and PR
 ↓
 Composition

Result: $+$ is P.R.

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

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

$a \times a \times \dots$
 $a \times 5$
 $= \underbrace{a + a + \dots + a}_5 \text{ times}$

$mul(a, i) = a;$

$\rightarrow f(n_1, 1) = n_1;$

$mul(\overset{n}{a}, \overset{m}{i+1}) = mul(\overset{n}{a}, \overset{m}{i}) + a \rightarrow f(n_1, m+1) = f(n_1, m) + n_1;$

$\therefore h(n_1, m, f) =$

In JAVA terms

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

mul = 0;

define, $n \cdot m = \underbrace{n + \dots + n}_{m \text{ times}}$

for (i=0; i < m; i++) {
 mul = mul + n;
 }

27/1/09
 Tuesday

mathematical term

$mul(n, m)$

$mul(n, 0) = 0$

$mul(n, m+1) = mul(n, m) + n$

General formula

$f(n_1, \dots, n_k, 0) = g(n_1, \dots, n_k)$

$f(n_1, \dots, n_k, m+1) = h(n_1, \dots, n_k, m, f(n_1, \dots, n_k, m))$

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

$f(n_1, m+1) = h(n_1, m, f(n_1, m))$

$f(n_1, 0) = 0$

$f(n_1, m+1) = f(n_1, m) + n_1$

$h(n_1, m, f) = \underbrace{f}_{3rd \text{ term}} + \underbrace{n_1}_{1st \text{ term}}$

elements = $\underbrace{\pi_3^3}_{\text{projecting term}} + \underbrace{\pi_1^3}_{\text{projecting term}}$

have $k+1$ terms

$mul = PR(0, add(\pi_3^3, \pi_1^3)) \rightarrow$ Since $f = PR(g, h)$
 how, $add = PR(\pi_1^3, \delta)$ here, $g = 0$
 $h =$

$= PR(0, [PR(\pi_1^3, \delta)](\pi_3^3, \pi_1^3))$

n^2 is PR.

$$0^2 = 0$$

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

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

$$\text{Square}(m+1) = \text{Square}(m) + m + m + 1$$

$$\therefore f(0) = 0$$

$$\therefore g = 0$$

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

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

$$= \pi_2^2 + \pi_1^2 + 6 \cdot \pi_1^2$$

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

is 1st term

is 2nd term

$$h = \text{add}(\pi_2^2, \text{add}(\pi_1^2, 6 \cdot \pi_1^2))$$

Fact(n) is P.R.

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

① for (i=0; i<n; i++)

$$\text{fact} = \text{fact} * i;$$

$$\rightarrow \text{fact}(i+1) = (i+1) * \text{fact}(i)$$

no of var.

here we have (i-1)=0 terms

$$[k=0]$$

② $\text{fact}(0) = 1$

③

$$\text{fact}(m+1) = \text{fact}(m) * (m+1) \rightarrow f(m+1) = f(m) * (m+1)$$

$$\rightarrow h(m, f) = f * (m+1)$$

$$= m \cdot f + f$$

$$* = \pi_1^2 \cdot \pi_2^2 + \pi_2^2 = \pi_2^2 (\pi_1^2 + 1)$$

$$= \pi_2^2 \cdot 6 \cdot \pi_1^2$$

Proof did

$$\text{fact} = 1;$$

for (i=1; i<n; i++)

$$\text{fact} = \text{fact} * i;$$

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

$$\text{fact}(m+1) = \text{fact}(m) * (m+1)$$

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

$$f(n_1, \dots, n_k, m+1) = h(n_1, \dots, n_k, m, f(n_1, \dots, n_k, m))$$

$$f(0) = g$$

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

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

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

$$g = 1$$

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

$$= m \cdot f + f$$

H.W.

③ Prove that, $n^3 - n^2 + 5$ is P.R.

① Prove that, a^n is P.R

② " " , $n^3 + n^2 + 5$ is P.R. ④ exclusive OR is PR

→ still needs to show -, --, /, %

$$a-b = \begin{cases} a-b & \text{if } a \geq b \\ 0 & \text{else} \end{cases}$$

Prev(3) = 2 let's prove, Prev is P.R.

Prev(2) = 1 prev(0) = 0 f(0) = g

Prev(1) = 0 prev(m+1) = m f(m+1) = h(m, f(m))

Prev(0) = 0

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

$$\therefore h = \Pi_1^2$$

$$\boxed{\text{Prev} = \text{PR}(0, \Pi_1^2)}$$

Now, PR $(0, \frac{g}{h}) \rightarrow$ Prove this is Prev function.

In general, $f(n_1, \dots, n_k, m) = g(n_1, \dots, n_k)$

$$f(n_1, \dots, n_k, m+1) = h(n_1, \dots, n_k, m, f(n_1, \dots, n_k))$$

In the f_0 , $k+2=2$ (h is fn of 2 variables)
 $\therefore k=0$

$$\left. \begin{aligned} f(0) &= g \\ f(m+1) &= h(m, f(m)) \end{aligned} \right\} \rightarrow \begin{aligned} f(0) &= 0 \\ f(m+1) &= m \end{aligned}$$

Π_1^2 (from the f_0)

In Java prog, prev = 0; for(i=0; i<n; i++)
prev = i;

Prove

$a-b$ is P.R.

Prog minus { = a;

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

for(i=0; i<b; i++)

minus = minus - 1;

$$\therefore \text{minus}(a, 0) = a$$

$$\text{minus}(a, m+1) = \text{prev}(a, m) [\text{minus}(a, m)]$$

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

$$\text{add}(m, m+1) = \text{add}(m, m) + 1$$

$$n \cdot (m+1) = n \cdot m + n$$

$$n - (m+1) = (n-m) - 1$$

'%' is P.R.

conditions: $\rightarrow <, >, =, <=, >=$, and, not.

Say $\begin{cases} \text{true} \equiv 1 \\ \text{false} \equiv 0 \end{cases}$

$$\text{positive}(0) = 0 \text{ (false)}$$

$$\text{positive}(m+1) = 1; m > 0$$

$$\therefore \text{PR}(0, 0, 0)$$

$\therefore m > 0$ is P.R.

$$a > b \quad a-b \text{ sh be +ve}$$

$$a > b \equiv \text{pos}(a-b) \quad \therefore \text{P.P.}$$

not is P.R.

$$\text{not}(0) = 1$$

$$\text{not}(0) = 1$$

$$\text{or, not}(m) = 1 - m$$

$$\text{not}(m+1) = 0$$

$$\text{not}(1) = 0$$

$$a \leq b \equiv \text{not}(a > b)$$

$$a < b \equiv \text{not}(b > a)$$

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

& is P.R (since & = mult.)

A	B	A B
0	0	0
0	1	1
1	0	1
1	1	1

De Morgan's theory

$$a \vee b = \neg(\neg a \wedge \neg b)$$

$$\text{here, } a + b - a * b \equiv A || B$$

$$a || b \equiv \text{not}(\text{mul}(\text{not}(a), \text{not}(b)))$$

$\therefore \vee$ is P.R, not &

$$a = b \equiv ! (a < b) \wedge ! (b < a) \quad \therefore \text{P.R.}$$

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

29/1/09
Tuesday

Def: A polynomial is any f_n that can be obtained from constants & variables by using +, *

e.g. $(n+1)(n-1)$ is a polynomial.

Exclusive OR: ^① $A \text{ XOR } B \equiv (A \vee B) \wedge \neg (A \wedge B)$ [A OR B but not both]
= and (OR(A,B), not (and(A,B)))

② $A \text{ XOR } B = \bar{A}B + B\bar{A} = (\neg A \wedge B) \vee (A \wedge \neg B)$

If-then. Statements:

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

if $(a > 0)$ then a else $-a$

let's assume: $h(n) = \text{if } P(n) \text{ then } f(n) \text{ else } g(n)$
 $\downarrow \qquad \qquad \qquad \downarrow \qquad \qquad \qquad \downarrow$
 P.R. \qquad \qquad \qquad P.R. \qquad \qquad \qquad P.R.

We want to prove $h(n)$ is P.R.

let's make it natural: let's do heuristic.

(either $P(n)$ and $f(n)$ is the value)

OR $(\neg P(n)$ and $g(n)$ is the value)

$\neg P = 1 - P$

$A \wedge B = A * B$

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

$\Rightarrow \boxed{P(n) * f(n) + ((1 - P(n)) * g(n))}$
 and OR not

h/w: this is P.R.
 ① Prove that

If $P(n)$ is true ($P=1$): $1 * f + 0 * g = f$

If P is false ($P=0$): $0 * f + 1 * g = g$

if $P_1(n)$ then $f(n)$
 else if $Q(n)$ then $g(n)$
 else $h(n)$

Remainder.

$0 \text{ rem } n = 0$

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

$0 \% 5 = 0$

$\text{rem}(n, m+1) = \text{if } (\text{rem}(n, m) + 1 < n) \text{ then } (\text{rem}(n, m) + 1) \text{ else } 0.$

[Java form: $(m \% n + 1 < n) ? (m \% n + 1) : 0$

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

\downarrow
 $\text{if } (\underbrace{f+1}_{\text{not}} < n) \text{ then } (\underbrace{f+1}_{\text{not}}) \text{ else } 0.$

Division

$$0/n = 0$$

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

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

$f(n, m+1) = \text{if } ((m+1) \% n = 0 \text{ then } (f+1) \text{ else } f.)$

Composition of P.R. functions, thus it is P.R.

In Java
Prog.

div = 0;

for(i=0; i < n; i++)

if ((i+1) % n == 0)
div++;

Is every computable fns P.R.?

2 Proofs. 1. in more details easier proof

2. in less detail. more natural.

Theorem: There exists a computable function which is not P.R.

P.R. f_2 in terms of $0, \sigma, \pi$, P.R.

writes

Writing with LaTeX!

it's like $\text{PR}(0, \sigma, \pi, \text{circle } \pi - 1^{\wedge} 1) \rightarrow \text{PR}(0, \sigma, \pi)$
↑
↓
central circle

seq of

0's & 1's

011 - - - -

how to describe it in natural number?

001

min last

Strip off front 1's to get the original seq's.

then find out the meaning of seq (which is sigma / circ)
then see whether the seqs are meaningful.

Example of

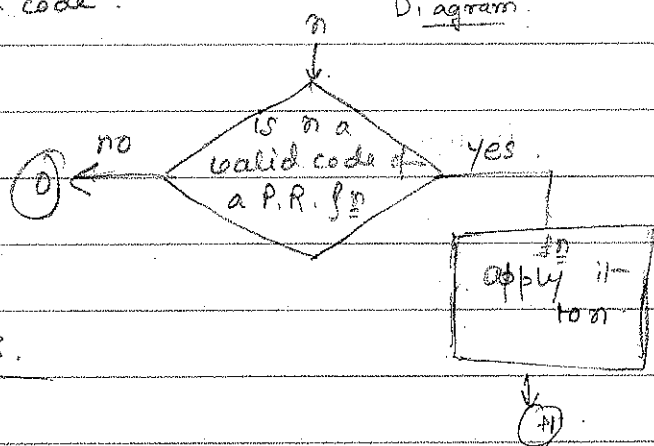
A f_n which is computable but not P.R.

then what?

mathematical description

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

Diagram



Statement: f is not P.R.

Proof by contradiction:

We assume, f be P.R. then f is represented by some expression,

that expression has a code c , $\forall x: f_c(x) = f(x)$

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

See the next pg

by def: $f(c) = f_c(c) + 1 \Rightarrow f(c) = f_c(c) + 1$

$$0 = 1$$

not true

then f is not P.R.

List all P.R. fns

$c \setminus n$	0	1	2	3	4	5	6	7
id. 0	0	1	2	3	4	5	6	7
Square 1	0	1	4	9	16	25	36	49
undefined 2	x	x	x	x	x	x	x	x
6. 3	1	2	3	4	5	6	7	8
Prav. 4	0	0	1	2	3	4	5	6
5								

$f(1), f(2), \dots$ all values are diff from the normal P.R. fns values.

We construct f_n which is all the

f [1 | 2 | 0 | 5 | 4 | | |] by applying

3/2/09
Tuesday

H.W. Reproduce the proof [computable but not P.R.]

Theorem: Not every computable f_n is P.R.

We will construct a function $f(m)$

which is * computable

* not P.R.

1st part. codes of P.R. f_n .

goal is to assign to every P.R. f_n a number.

We start with an expression. $PR(\pi_i, 600)$

making diff int
by putting 1 in front
[11 = 3
101 = 5
1001 = 9]

\downarrow (latex)
 $PR(\backslash pi^i - 1, \backslash sigma \backslash circ 0) \rightarrow$ Typeable form
 \downarrow ASCII (when type in comp. only seq of 0's & 1's)

Put 1 in front then every seq. will be represented on diff int.

0110
10110
 \downarrow
Code of a P.R. f_n

Thus. computable.

P.R. compiler checks that.

statement This is an algorithm, that given a natural number C

- * checks whether C is a valid code of a P.R. f_n
- * if it is produces a Java code f_c for computing the corresponding P.R. f_n . This Java code is denoted f_c

Idea ① strip off the first "1"

② check if all combinations are ASCII symbols.

③ Latex compiler checks whether syntax is correct.

④ check syntax.

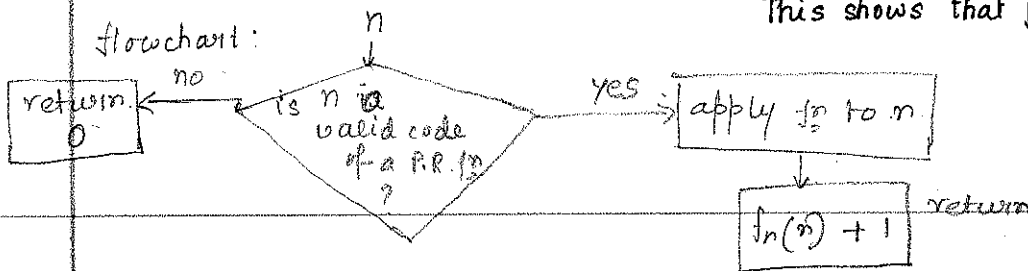
$PR(\hat{f}, \hat{g})$
 $\downarrow \downarrow \downarrow \downarrow$
checks for whether $PR, (,),$ are there

then transforms it to $h = f$.

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

$h = 9$

This shows that f is computable.



then $f(n) = *$ (see prev page)

** [since c is a valid code, by def of f , we have;

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

in particular, for $x = c$; $f_c(c) = f(c)$]

P.R. but not computable. $f(n) \stackrel{\text{def}}{=} \begin{cases} f_n(n) + 1 & \text{if } n \text{ is a valid code of a P.R. } f_n \\ n + 1 \neq 0 & \text{otherwise} \end{cases}$

Georg Cantor (1845-1918) invented set theory.

Real numbers can't be enumerated. (listed)

All natural nos: $\{0, 1, 2, \dots, n, \dots\}$

All integers: $\{0, 1, -1, 2, -2, 3, -3, \dots\}$

All fractions: $\{0, 1/1, 1/2, 2/1, 2/2, 1/3, 2/3, \dots\}$

Now listing Real nos.		0	1	2	3	4	5	6
$1/3$ x_0	0	3	3	3				
$\sqrt{2}$ x_1	1	4	1	2				
$\sqrt{3}$ x_2	3	1	4	1				
\vdots	\vdots							
x_n	2	7	8	1	8	2	8	

Cantor's proof: we only care about

the existence.

PR proof: we also care about computable

diagonalization

f [1 | 5 | 5 | ... 9] → This is a construction of Real no which doesn't \in to the seq.

Result \exists a computable f_n which is not P.R.

1st Proof: easier to describe, but f is meaningless

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

Reminder we started with 6.

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

level 1: $f_1(n, f_0(n, \dots))$

add $f_1(n, m+1) = f_0(f_1(n, m), n) \leftarrow$

$add(n, m+1) = add(n, m) + 1$

mul $f_2(n, m+1) = f_1(f_2(n, m), n) \leftarrow$

$mul(n, m+1) = mul(n, m) * n$

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

$n^{m+1} = n^m * n \rightarrow pow(n, m+1) = pow(n, m) * n$

$f_{k+1}(n, m+1) = f_k(f_{k+1}(n, m), n) \rightarrow$ Archimedes.

Next times. $(a^a)^a = a^{a^a}$

start from here $(2^2)^2 = 2^4$

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

$4^4 = 256$
 $4^4 = 4 = 2 = 2 \approx 10^{1.53}$

Ackermann

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

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

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

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

$A(4) = f_4(4, 4) = 4^{4^4} = 4^{(10^{1.53})^4}$

$2^{10} \approx 10^3$ $\frac{512 \cdot 3}{10} = 153$

Intro to the theory of computation
 Michael Sipser

Tuesday
 A research topic

Ackermann's function

Idea of proving that $A(n)$ is not P.R.

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

P.R. f has for loops.

So, with no for-loops, f constructed

- $0, 0, \pi_i^k, 0$
- $0, 0(0), 0(0(0)) \dots$
- $\pi_1, 0(\pi_1), \dots$
- $n, n+1, n+2, \dots$

(constant) \rightarrow bounded by
 So, $\exists C$ such that, $f(n) \leq n + C, f(n_1, \dots, n_k) \leq n_i + C$

not every computable f is A.P.R.

Let's take no variable f_n ,

$$f(0) = g \quad \text{bounded by}$$
$$f(m+1) = h(m, f(m)) \leq \max(m, f(m)) + C$$

Since, $f(m+1)$ is not P.R., means there is no for loop
thus, $f(m)$ cannot be a P.R. thus bounded by $*$

$$f(0) = g$$
$$f(1) \leq \max(1, g) + C \leq \tilde{g} + C \quad \tilde{g} \text{ is } \max(1, g)$$
$$f(2) \leq \max(2, \tilde{g} + C) + C \leq \tilde{g} + 2C$$

⋮

$$f(m) \leq \tilde{g} + m * C$$

for no for-loop: $f(n) \leq n + \underbrace{\dots + 1}_{\text{const. times}}$

for one for-loop

$$\therefore f(m+1) \leq c f(m)$$

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

Ackerman f_n grows faster than any f_n .

$$f(0) = (n+1) \quad \text{grows faster than prev. step}$$

$$f(1) = n + n = (2n)$$

$$f(2) = (n^2)$$

And any P.R. f_n is somewhere
in b/w this levels.

⇒ Our 1st hypothesis: every computable f_n is P.R.

fact There is one computable f_n which is not P.R.

Example. Ackermann's f_n . $A(n)$

Natural idea: A P.R. f_n is anything that can be
obtained from $0, \pi_i^k, \delta$, and $A(n)$ by \circ and P.R.
a basic f_n .

Q. Is every computable f_n A-P.R.?

what is

List of all computable f_n

Missing: while-loop

We need to formalize while-loop.

while (!P)
{ }

No of iteration \equiv smallest m for which $P(\vec{n}, m)$ is true

μ -recursion

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

smallest (when the condition fails.)

A μ -recursive f_n is any f_n which can be obtained from 0, δ , π_i^k by σ , PR, and μ -recursion.

$$\lfloor a - b \rfloor = 0; a < b. \quad a - b = c / c + b = a.$$

$$3 - 5 = 0 = a - b; a > b$$

$$5 - 3 = 2$$

$$3 - 5$$

$$2 - 5$$

$$\mu c (c + 5 \geq 2)$$

\downarrow
we stop at $c = 0$

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

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

$$\mu c (c + 2 \geq 5)$$

$5 - 2 = 3$ here we stop at $c = 3$

H.W \Rightarrow Prove $/$ is μ -recursive f_n

a/b

$$5/2 = 2$$

$$2/5 = 0$$

Infinite Loop

$f(n) = \mu m (0 = 1)$. $m = 0$; while (! (0 = 1)) { m++ } This f_n is never defined.

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

$\mu m (n = 0 \ \& \ m = 0)$ if $n = 0 \rightarrow$ return 0
 $n \neq 0 \rightarrow$ undefined

here $n = 0$ then return 0

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

$$\text{um} [(n=2 \ \& \ m=1) \vee (n=3 \ \& \ m=2)]$$

if $n=2$: [returns 1]

if $n=3$: [1, 2]

H.W

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

μ -recursive fns that may be undefined sometimes.

totally recursive \equiv μ -recursive and everywhere defined.

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

Imp. In 1930, 2 diff. definitions appeared.

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

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

Turing
myc.

Tuesday
10.02.09

Computable \equiv μ -recursive.

Big problem with while-loops:

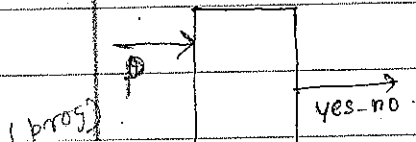
may not stop. (goes into ∞ loop.)

(halt)

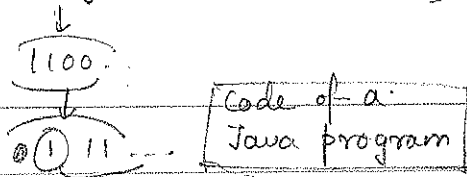
Theorem: No algorithm is possible that given a program p & data d , would check whether p halts on d (or not)

we would like to have.

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



Let's take P, one program
P in Java (It's in ASCII)

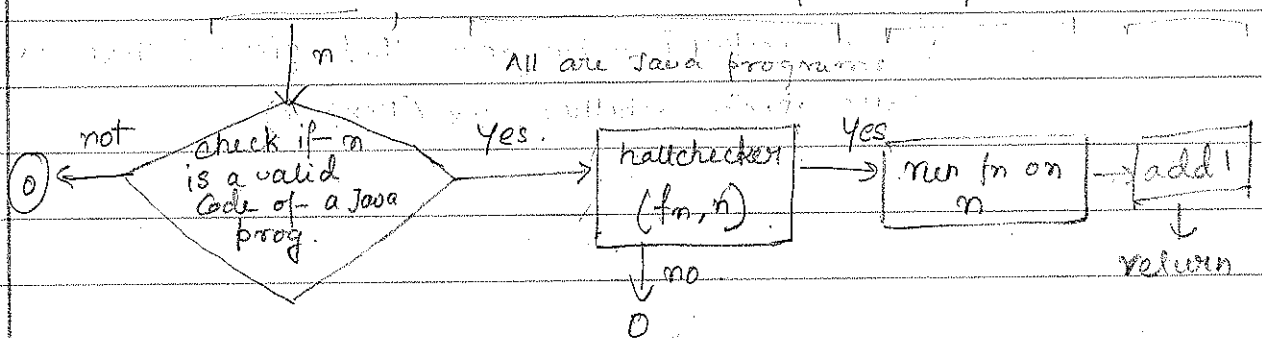


Reduction to contradiction, let's assume that halt-checker exists

let's define,

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

We start with 'n' → then Java compiler decides whether n is syntactically correct.



f is computable by some Java prog, let's denote by C

the code of this Java prog. $\forall x (f_c(x) = f(x))$

By def. of f, since C is a valid code & f_c always halts. In particular for $x = C$:

$$f_c(C) = f(C) \quad \dots \textcircled{2}$$

$$f_c = f_c(C) + 1 \quad \dots \textcircled{1}$$

from ① & ② Two #'s $f_c(C)$ & $f_c(C) + 1$ are equal to

the same # $f(C)$, so they must be equal to each other.

$$f_c(C) = f_c(C) + 1$$

$$\therefore 0 = 1$$

contradicts our assumption.

Diagonalization is relativized.

h.w. 1. P.R. proof if needed.

2. Reproduce the halting proof.

3. " z.e proof

4. Prove that cube checkers are impossible.

A program: ① does it halt?



algo. undecidable

② If it halts, does it produce the correct result?

We want the fn, $\forall x (f(x)=0)$.

In Java, `public static int zero(int x) { return x - x; }`

but, `return x/x - 1;` is not always 0 when $x=0$, it's undefined.

Def. A zero-checker is a program 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: zerochecker do not exist.

Proof by contradiction.

We assume that zerochecker exists.

↳ we will conclude that a halchecker exists.

We want:

$halchecker(p, d) = \begin{cases} 1 & \text{if } p \text{ halts on } d \\ 0 & \text{if } p \text{ doesn't halt on } d \end{cases}$

We can design:

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

↳ computable.

$P \text{ halts on } d \equiv \exists t \text{ } halt(p, d, t)$

$P \text{ doesn't halt on } d \equiv \forall t \text{ } /halt(p, d, t) = 0$

