MATHEMATICS FOR INFORMATION SCIENCE
NO.3 RECURSIVE FUNCTION

Tatsuya Hagino
hagino@sfc.keio.ac.jp

Slides URL

https://vu5.sfc.keio.ac.jp/slide/

So far

- Computability
  - While program and flow chart are equivalent.

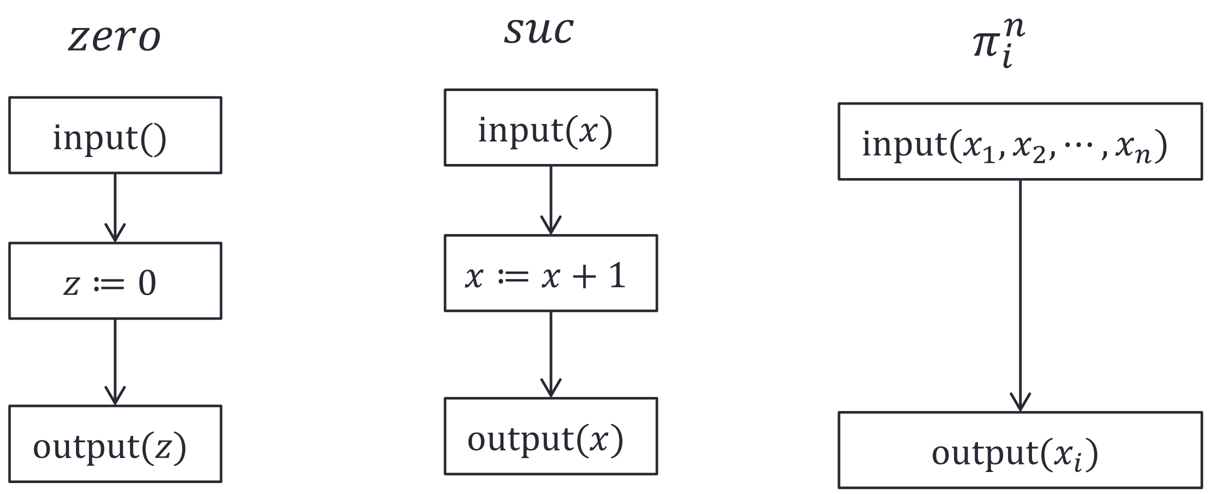
- Primitive recursive function
  - $zero : N^0 \to N$ $zero() = 0$
  - $suc : N \to N$ $suc(x) = x + 1$
  - $\pi_i^n : N^n \to N$ $\pi_i^n(x_1, \ldots, x_n) = x_i$
  - primitive recursion
    - $f(x_1, \ldots, x_n, zero()) = g(x_1, \ldots, x_n)$
    - $f\big(x_1, \ldots, x_n, suc(y)\big) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$
  - composition of primitive recursive functions
    - $f(x_1,, \ldots, x_n) = g(h_1(x_1,, \ldots, x_n), \ldots, h_m(x_1,, \ldots, x_n))$

- Example of primitive recursive functions:
  - one, pred, add, sub, mul, div, ...

Compute Primitive Recursive Functions

- **Theorem:**
  - Primitive recursive functions are computable.

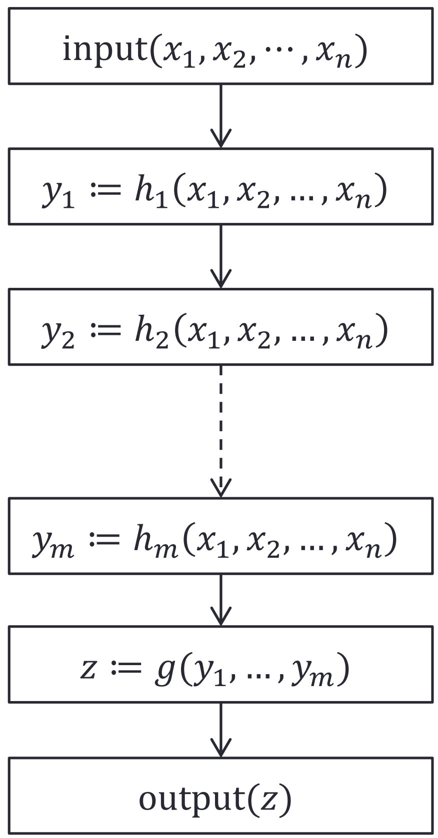
- **Proof:**
  - $zero, suc, \pi_i^n$ are computable.



Compute Primitive Recursive Functions

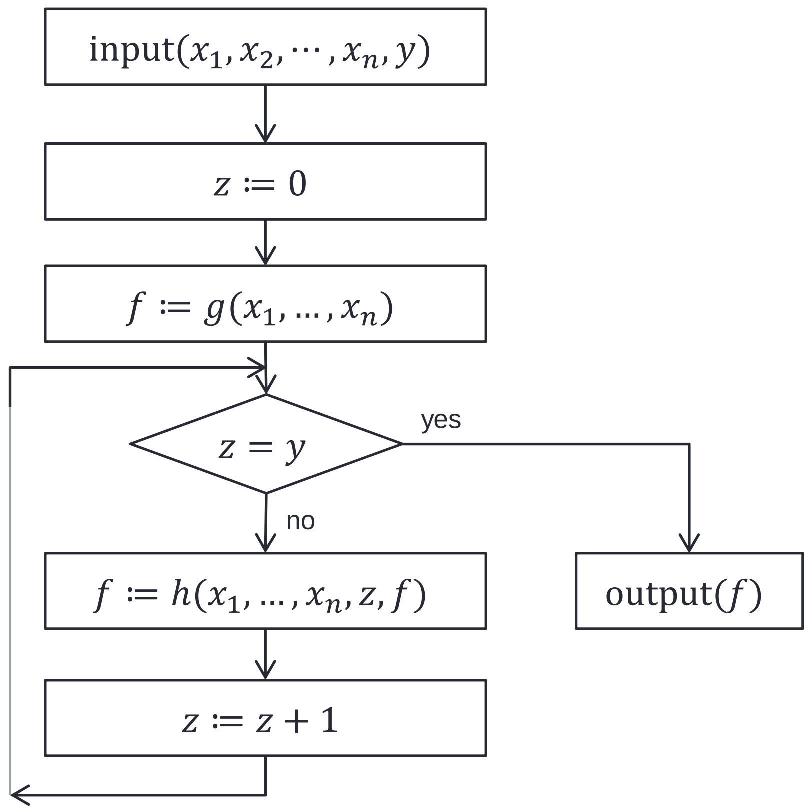
- A composition of primitive recursive functions are computable.

$$f(x_1, x_2, \dots, x_n) = g\big(h_1(x_1, x_2, \dots, x_n), \dots, h_m(x_1, x_2, \dots, x_n)\big)$$



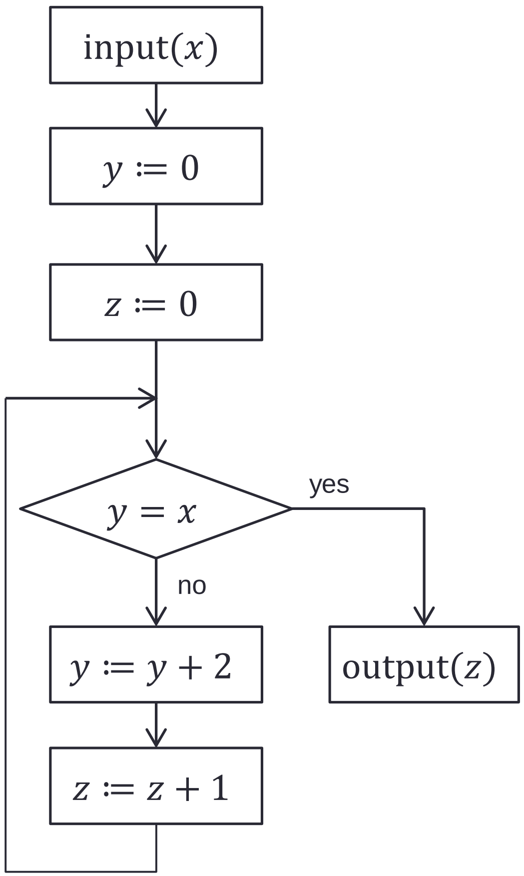
Compute Primitive Recursive Functions

- A function defined by a primitive recursion is computable.
  - $f(x_1, \ldots, x_n, zero()) = g(x_1, \ldots, x_n)$
  - $f\big(x_1, \ldots, x_n, suc(y)\big) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$



Is computable function always primitive recursive?

- Primitive recursive functions are total.
  - total = for any input, there is output.
- Computable functions may not be total, but partial.
  - partial = for some input, there is no output.
- The set of computable functions is larger than that of primitive recursive functions.
- There is a total function which is not primitive recursive:
  - Ackerman function $A: N^2 \to N$
    - $A(0, y) = suc(y)$
    - $A(suc(x), 0) = A(x, suc(0))$
    - $A(suc(x), suc(y)) = A(x, A(suc(x), y))$



Minimization Operator

- Definition:
  - For predicate $p: N^{n+1} \rightarrow \{\text{True}, \text{False}\}$

$$f(x_1, \dots, x_n) = \min(\{y \mid p(x_1, \dots, x_n, y) \text{ is True}\})$$

- $f(x_1, \dots, x_n)$ gives the smallest y which makes $p(x_1, \dots, x_n, y)$ true.
  - $f(x_1, \dots, x_n)$ is called minimization function of $p(x_1, \dots, x_n, y)$ and is written as:

$$\mu_y\big(p(x_1, \dots, x_n, y)\big)$$

  - μ is know as minimization operator.

- Example:
  - $f(x) = \mu_y(x = y \times 2)$ $\quad f(2) = 1 \quad f(3) = \perp$
  - $g(x) = \mu_y(x = y^2)$ $\quad g(4) = 2 \quad g(5) = \perp$

Recursive Function

- Recursive Functions:
  - Primitive recursive functions
  - Minimization functions for primitive recursive predicates
  - Composition of recursive functions
  - Functions defined by primitive recursion with recursive functions

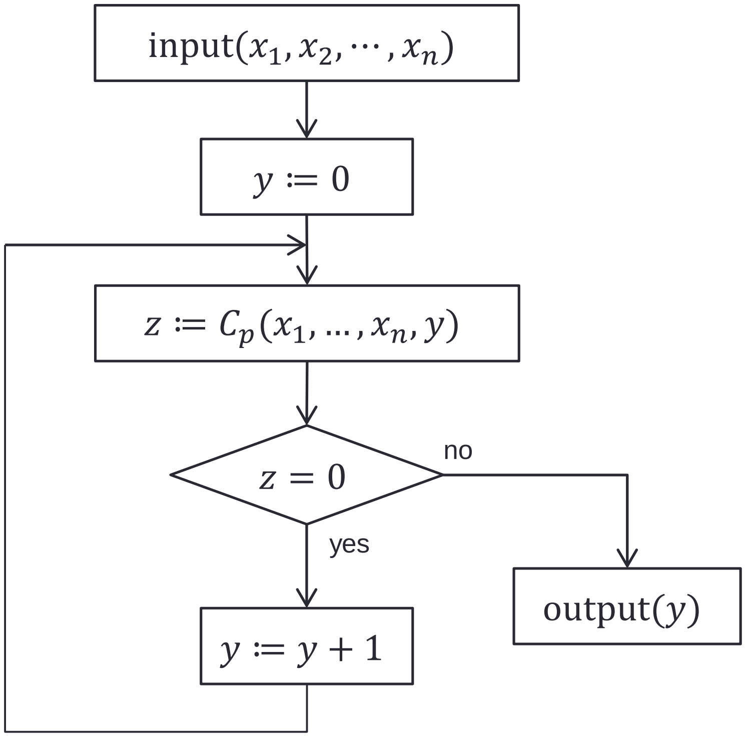
- In short, recursive function is:
  - primitive recursive function + minimization operator

Recursive ⇒ Computable

- **Theorem:** Recursive functions are computable.

- **Proof:**
  - Only need to show about the minimization operator.

$$f(x_1, \dots, x_n) = \mu_y\big(p(x_1, \dots, x_n, y)\big)$$



Gödel Function

- Gödel function $G: N^n \rightarrow N$ and its inverse functions $G_1: N \rightarrow N, \dots, G_n: N \rightarrow N$ must satisfy:
  - G is a one-to-one function,
  - $G_i\left(G(x_1, \dots, x_n)\right) = x_i$, and
  - $G, G_1, \dots, G_n$ are primitive recursive.

- $G(x_1, \dots, x_n)$ is called Gödel number of $x_1, \dots, x_n$.

- Example:
  - $G(x_1, x_2, \dots, x_n) = 2^{x_1} \times 3^{x_2} \times \cdots \times p_n^{x_n}$ (where p_n is the nth prime number)
  - $G_1(x) = x - \mu_{y<x}\left(\text{divisible}(x, 2^{x-y})\right)$
  - $G_2(x) = x - \mu_{y<x}\left(\text{divisible}(x, 3^{x-y})\right)$

$$\vdots$$

  - $G_n(x) = x - \mu_{y<x}\left(\text{divisible}\left(x, p_n^{x-y}\right)\right)$

Computable ⇒ Recursive

- **Theorem:** Computable functions are recursive.

- **Proof:**
  - Any while program can be converted into the following format:

input($x_1, \ldots, x_n$);
$a \coloneqq 1$;
while ($a - k = 0$) {
    if ($a = 1$) P_1;
    else if ($a = 2$) P_2;
    else if ($a = 3$) P_3;
      ⋮
    else if ($a = k$) P_k;
}
output(y)

where P_i is either an assignment or a conditional statement.

Proof (cont.)

- Let $a_1, \dots, a_n$ be all the variables in the program.
  - Let a_1 be the box number variable a.
  - Use $b = G(a_1, \dots, a_n)$ instead of individual variables.

- If P_i is an assignment statement: $a_m \coloneqq f(a_1, \dots, a_n); a \coloneqq l$
  - $b \coloneqq G\left(l, G_2(b), \dots, f\left(G_1(b), \dots, G_n(b)\right), \dots, G_n(b)\right)$

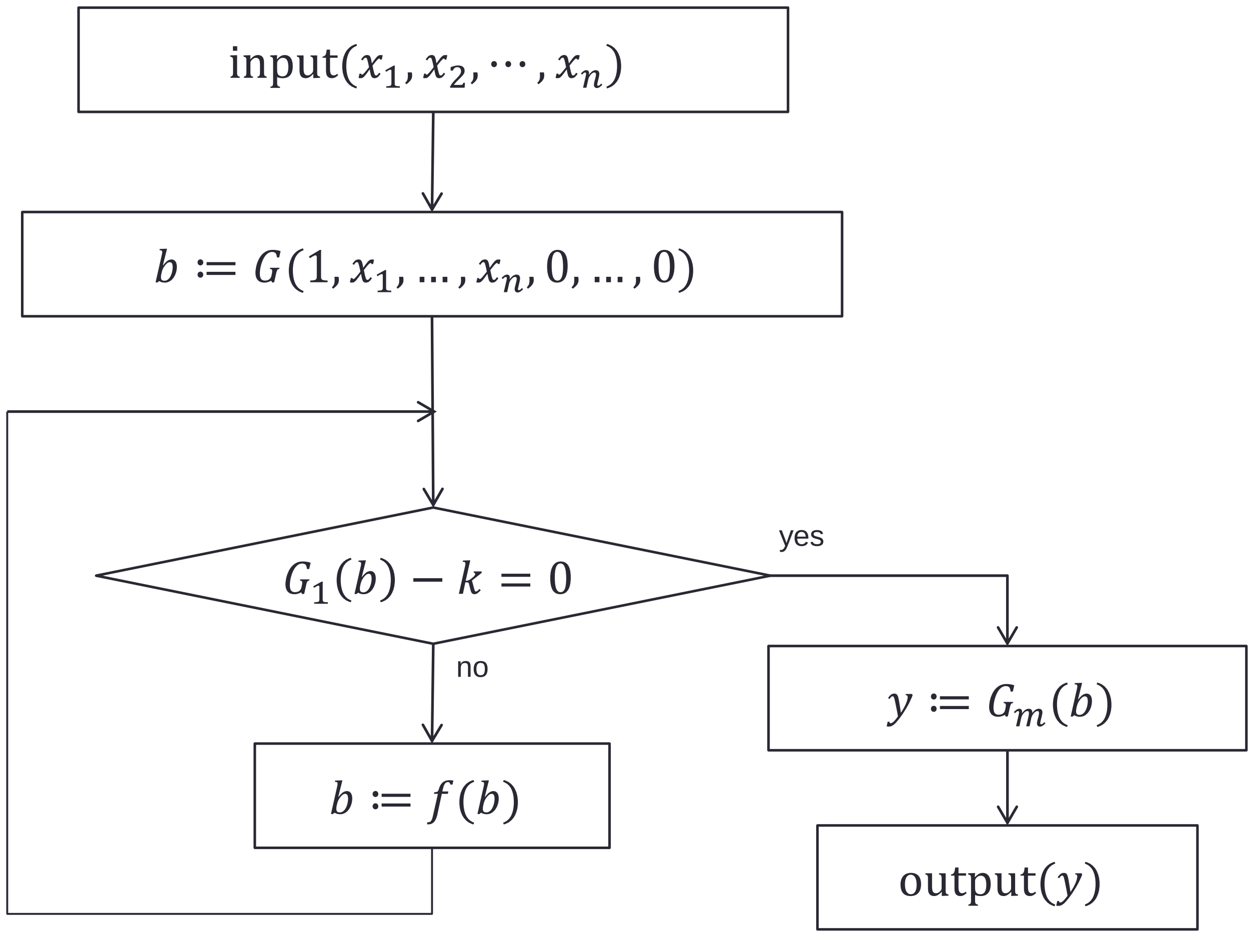
- If P_i is a conditional statement: if $(p(a_1, \dots, a_n))$ $a \coloneqq l$ else $a \coloneqq m$
  - $b \coloneqq G\left(C_p\left(G_1(b), \dots, G_n(b)\right) \times l + \left(1 - C_p\left(G_1(b), \dots, G_n(b)\right)\right) \times m, G_2(b), \dots, G_n(b)\right)$

- P_i can be expressed as a simple assignment statement
  - $b \coloneqq f_i(b)$
  - where f_i is a primitive recursive function.

- Combining section of P_i depending on a can also be expressed as a single assignment statement:
  - $b \coloneqq \sum_{i=1}^{k} C_=(G_1(b), i) \times f_i(b)$

Proof (cont.)

- The program can be converted into the following:



Proof (cont.)

- Let $f^{\#}(b,n) = f\left(f\left(f(\cdots f(b))\right)\right)$
  - Apply f to b n times.
  - Can be defined by primitive recursion:
    - $f^{\#}(b,0) = b$
    - $f^{\#}(b, suc(n)) = f\left(f^{\#}(b,n)\right)$
- The loop can be expressed using minimization operator.
  - $h(b) = f^{\#}\left(b, \mu_n\left(G_1\left(f^{\#}(b,n)\right) > k\right)\right)$
- Therefore, the program calculates the following function:
  - $G_m\left(h(G(1, x_1, \dots, x_n, 0, \dots, 0))\right)$
- This is a recursive function. (QED)

Lemma

Any recursive function can be expressed as

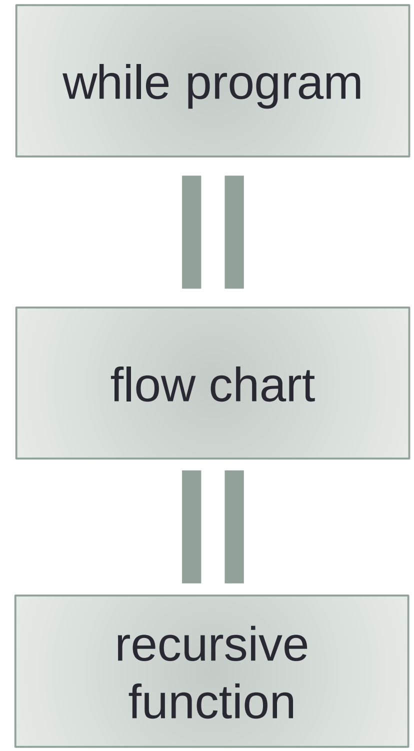
$$f\left(x_1, \dots, x_n, \mu_y\left(p(x_1, \dots, x_n, y)\right)\right)$$

where f is a primitive recursive function and p is a primitive recursive predicate.

- only one μ is necessary
- others are primitive recursive

Summary

- Primitive recursive functions:
  - Summation and Product
  - Primitive recursive predicate
  - division is primitive recursive
  - nth prime number is primitive recursive
- Recursive functions:
  - Primitive recursive functions
  - Minimization operator
- Any recursive function is computable.
- Any computable function is recursive.



Mathematical Induction

- In order to show $P(x)$ holds for any natural number x, show the following two things:
  - (base) It holds for $x = 0$
  - (induction) Assuming it holds for $x = n$, it also holds $x = suc(n)$

- This is called mathematical induction.
  - $P(x)$ holds for natural number x by mathematical induction.

$$\frac{P(0) \qquad P(n) \supset P\big(suc(n)\big)}{\forall x \in N \;\; P(x)}$$

Show $add(0, x) = x$

Lemma: $add(0, x) = x$

Definition of add
- $add(x, 0) = x$
- $add(x, suc(y)) = suc(add(x, y))$

- Proof:

(base) If $x = 0$, from the definition $add(0,0) = 0$. Therefore, it holds.

(induction) Assume it holds for $x = n$. Then $add(0, n) = n$.

If $x = suc(n)$,

$lhs = add(0, suc(n))$

$= suc(add(0, n))$ ($\because$ definition of add)

$= suc(n)$ ($\because$ assumtion)

$= rhs$

Therefore $add(0, x) = x$ holds for any natural number x.

Homework: Prove $add(x, y) = add(y, x)$

Theorem: $add(x, y) = add(y, x)$

- Before proving this theorem, you may need to prove the following lemma.

Lemma: $add(suc(x), y) = suc(add(x, y))$

- This lemma can be proved by mathematical induction on y.
- Then, you can prove the theorem by mathematical induction on x.