

Computability

1.1 Foundation of Recursive Function Theory

computation

operational approach

how a computation is performed

functional approach

what a computation accomplishes

computable

some function rather than

some algorithm in a particular system

functional approach

function computed

means of computing them

the study of recursive function theory

initial functions

combined functions

Partial Functions

$f: \mathbf{N}^m \rightharpoonup \mathbf{N}^n$ where $m, n \in \mathbf{N} (m, n \geq 0)$

various domain and range

*any data can be coded into 0's and 1's
tuples of nonnegative integers*

Ex. $\text{div}(x, y) = x/y$, if $y \neq 0$.

partial function

strictly partial function

total function

arbitrary m -tuple $\bar{x} = (x_1, \dots, x_m)$

Three Initial Functions

1. zero function, $\zeta: \mathbf{N}^m \rightarrow \{0\} (0 \leq m)$

$$\zeta(\bar{x}) = 0$$

2. successor function, $\sigma: \mathbf{N} \rightarrow \mathbf{N}$

$$\sigma(x) = x + 1$$

3. projections, $\pi_n^m: \mathbf{N}^m \rightarrow \mathbf{N} (0 \leq n \leq m)$

$$\pi_n^m(x_1, \dots, x_m) = x_n.$$

all of three initial functions are computable.

Primitive Recursive Functions

1. combination of functions: \times

Define $f \times g: \mathbf{N}^m \rightarrow \mathbf{N}^{n+k}$ from

$f: \mathbf{N}^m \rightarrow \mathbf{N}^n$ and $g: \mathbf{N}^m \rightarrow \mathbf{N}^k$ by

$$f \times g (\bar{x}) = (f(\bar{x}), g(\bar{x})).$$

2. composition of functions: \circ

Define $f \circ g: \mathbf{N}^m \rightarrow \mathbf{N}^k$ from

$f: \mathbf{N}^m \rightarrow \mathbf{N}^n$ and $g: \mathbf{N}^n \rightarrow \mathbf{N}^k$ by

$$f \circ g (\bar{x}) = g(f(\bar{x})).$$

Ex): $\zeta \circ \sigma(\bar{x}) = 1$

3. primitive recursion of functions: f

Define $f: \mathbf{N}^m \rightarrow \mathbf{N}^k$ from

$g: \mathbf{N}^{m-1} \rightarrow \mathbf{N}^k$ and $h: \mathbf{N}^{m+k} \rightarrow \mathbf{N}^k$ by

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

$$f(\bar{x}, y+1) = h(\bar{x}, y, f(\bar{x}, y))$$

where $\bar{x} \in \mathbf{N}^{m-1}$ and $y \in \mathbf{N}$.

or

$$f(\bar{x}, \zeta(y)) = g(\bar{x})$$

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

where $\bar{x} \in \mathbf{N}^{m-1}$ and $y \in \mathbf{N}$.

Example:

$f(x, y)$: the number of nodes
in a full x -nary balanced tree whose depth is y

$$f: \mathbf{N}^2 \rightarrow \mathbf{N}$$

level y : x^y nodes

For fixed x , $f(x, y+1) = f(x, y) + x^{y+1} = f(x, y) + x^y \cdot x$

$$f(x, 0) = x^0 \quad (1)$$

$$f(x, y+1) = f(x, y) + x^y \cdot x \quad (2)$$

f is **computable**

$$\begin{aligned} f(3, 2) &= f(3, 1) + 3^1 \cdot 3 &&= f(3, 1) + 9 \\ &= f(3, 0) + 3^0 \cdot 3 + 9 &&= f(3, 0) + 12 \\ &= 3^0 + 12 = 13 \end{aligned}$$

$$f(x, 0) = x^0$$

$$f(x, y+1) = f(x, y) + x^y \cdot x$$

or

Let $g: \mathbf{N} \rightarrow \mathbf{N}$ and $h: \mathbf{N}^3 \rightarrow \mathbf{N}$

where $g(x) = x^0$ and $h(x, y, z) = z + x^y \cdot x$

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

$$f(x, y+1) = h(x, y, f(x, y))$$

Primitive recursion

Let $g: \mathbf{N}^{m-1} \rightarrow \mathbf{N}^k$ and $h: \mathbf{N}^{m+k} \rightarrow \mathbf{N}^k$. Then

$f: \mathbf{N}^m \rightarrow \mathbf{N}^k$ ($1 \leq m$, $0 \leq k$) is defined by

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

$$f(\bar{x}, y+1) = h(\bar{x}, y, f(\bar{x}, y))$$

where \bar{x} denotes $(m-1)$ -nary vector

f is ***primitive recursion*** of g and h , if

$$f(\bar{x}, \zeta()) = g(\bar{x})$$

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

primitive recursion is computable

$$\begin{aligned} \underline{f}(\bar{x}, 3) &= h(\bar{x}, 2, \underline{f}(\bar{x}, 2)) \\ &= h(\bar{x}, 2, h(\bar{x}, 1, \underline{f}(\bar{x}, 1))) \\ &= h(\bar{x}, 2, h(\bar{x}, 1, h(\bar{x}, 0, \underline{f}(\bar{x}, 0)))) \\ &= h(\bar{x}, 2, h(\bar{x}, 1, h(\bar{x}, 0, g(\bar{x})))) \end{aligned}$$

primitive recursive functions

constructed from ***initial functions***
with ***finite number of***
combinations, compositions,
and primitive recursions

Every primitive recursive function is total and computable.

initial functions are total and computable.

finite number of \times , \circ , and primitive recursions applied to total and computable.

\therefore Primitive recursive functions are total and computable.

primitive recursive functions contains most, if not all, of computable total functions (Ackermann function) that are required in traditional computer applications

But every computable function is not primitive recursive.

There are computable functions that are not primitive recursive.

Ackermann function is total and computable but not primitive recursive.

Division is partial

primitive recursive \subset computable total functions \subset computable functions

1.2 The Scope of Primitive Recursive Functions

1. extend primitive recursive functions

includes total computable functions

in typical computer applications

provide specific examples (following sections)

2. distinction between primitive recursive functions and computable total functions

Consider

$$(\pi_1^3 \times \pi_3^3) \circ \text{plus}(x, y, z) = \text{plus}(x, z) = x + z$$

Examples of primitive recursive functions

constant functions

$$K_n^m: \mathbf{N}^m \rightarrow \{n\}, m, n \in \mathbf{N}.$$

$$\text{where } K_n^m(\bar{x}) = n, \bar{x} \in \mathbf{N}^m.$$

For $m = 0$

$$K_2^0 = \zeta \circ \sigma \circ \sigma = 2$$

$$K_n^0 = \zeta \circ \sigma \circ \dots \circ \sigma = n$$

$\therefore K_n^0$ is primitive recursive

For $m > 0$,

$$K_n^m(\bar{x}, 0) = K_n^{m-1}(\bar{x})$$

$$K_n^m(\bar{x}, \sigma(y)) = \pi_{m+2}^{m+2}(\bar{x}, y, K_n^m(\bar{x}, y)) = K_n^m(\bar{x}, y)$$

$\therefore K_n^m$ is primitive recursive ($n \geq 0$).

$$\begin{aligned}
& K_n^m(n_1, \dots, n_m) = K_n^m(n_1, \dots, n_{m-1}) = \dots \\
& = K_n^m(n_1, \dots, n_{m-1}, 0) = K_n^{m-1}(n_1, \dots, n_{m-1}) \\
& = \dots \\
& = K_n^1(n_1) = \dots = K_m^1(0) \\
& = K_n^0() = \zeta \circ \sigma \circ \dots \circ \sigma () = \dots = n
\end{aligned}$$

Consider

$$C_{n_1 \dots n_k}^{m,k}: \mathbf{N}^m \rightarrow \mathbf{N}^k \quad K_{n_1}^m \times \dots \times K_{n_k}^m: k \text{ times}$$

$$C_{n_1 \dots n_k}^{m,k}(\vec{x}) = (n_1, \dots, n_k)$$

Constants are also primitive recursive functions

$$7 = K_7^n$$

$$(2, 5) = K_2^n \times K_5^n$$

arithmetic functions

$$\text{mult}(x, 0) = K_0^1(x)$$

$$\text{mult}(x, \sigma(y)) = (\pi_1^3 + \pi_3^3) \circ \text{plus}(x, y, \text{mult}(x, y))$$

$$\text{mult}(x, 0) = 0$$

$$\text{mult}(x, y+1) = \text{plus}(x, \text{mult}(x, y))$$

$$\text{expo}(x, 0) = K_1^1(x)$$

$$\text{expo}(x, \sigma(y)) = (\pi_1^3 \times \pi_3^3) \circ \text{mult}(x, y, \text{expo}(x, y))$$

$$\text{expo}(x, 0) = 1$$

$$\text{expo}(x, y+1) = \text{mult}(x, \text{expo}(x, y))$$

predecessor function

$$\text{pred}: \mathbf{N} \times \mathbf{N}$$

$$\text{pred}(0) = \zeta() = 0$$

$$\text{pred}(\sigma(x)) = \pi_1^2(x, \text{pred}(x)) = x$$

$$\text{monus}(x, 0) = \pi_1^1(x)$$

$$\text{monus}(x, \sigma(y)) = \text{pred} \circ \pi_3^3(x, y, \text{monus}(x, y))$$

$$\text{monus}(x, 0) = x$$

$$\text{monus}(x, y+1) = \text{pred}(\text{monus}(x, y))$$

$$\text{monus} \equiv \dot{-}$$

$$x \dot{-} y = x - y, \text{ if } x \geq y; 0, \text{ otherwise.}$$

$$\text{eq}: \mathbf{N}^2 \rightarrow \mathbf{N}$$

$$\text{eq}(x, y) = 1 \text{ if } x = y, 0 \text{ if } x \neq y$$

$$\text{eq}(x, y) = 1 \dot{-} ((x \dot{-} y) + (y \dot{-} x))$$

$$= \text{monus}(1, \text{plus}(\text{monus}(x, y), \text{monus}(y, x)))$$

Example:

$$\text{eq}(5, 3) = 1 \dot{-} ((5 \dot{-} 3) + (3 \dot{-} 5))$$

$$= 1 \dot{-} (2 + 0) = 1 \dot{-} 2 = 0$$

$$\text{eq}(5, 5) = 1 \dot{-} ((5 \dot{-} 5) + (5 \dot{-} 5))$$

$$= 1 \dot{-} (0 + 0) = 1 \dot{-} 0 = 1$$

Beyond Primitive Recursive Functions

initial functions

\subset *primitive recursive functions*

\subset *computable functions*

computable ftns $\not\subset$ *primitive recursive functions*

div(partial function)

primitive recursive functions $? \subset$ *computable total*

Ackermann function (W. Ackermann in 1928)

$A: \mathbb{N}^2 \rightarrow \mathbb{N}$

$$A(0, y) = y + 1$$

$$A(x+1, 0) = A(x, 1)$$

$$A(x+1, y+1) = A(x, A(x+1, y))$$

**Ackermann function is *computable total* (App. B)
but *not primitive recursive*.**

$$\begin{aligned} A(3, 2) &= A(2, A(3, 1)) \\ &= A(2, A(2, A(3, 0))) \\ &= A(2, A(2, A(2, 1))) \\ &= A(2, A(2, A(1, A(2, 0)))) \\ &= A(2, A(2, A(1, A(1, 1)))) \\ &= A(2, A(2, A(1, A(0, A(1, 0)))) \\ &= A(2, A(2, A(1, A(0, A(0, 1)))) \\ &= A(2, A(2, A(1, A(0, \underline{2})))) \\ &= \dots \end{aligned}$$

Theorem 4.1 *There is a computable total function from \mathbf{N} to \mathbf{N} that is not primitive recursive.*

Proof

the number of p.r. functions is countable
(finite combination, composition, p. rec.)

$\therefore f_1, \dots, f_n, \dots$

$f: \mathbf{N} \rightarrow \mathbf{N} \ .\exists. f(n) = f_n(n) + 1$ *computable total*

If f is p.r., $\exists m \in \mathbf{N} \ .\exists. f = f_m$. But $f(m) = f_m + 1$

$\therefore f$ *is computable but not p.r.*

(\therefore no. of computable functions is uncountable)

computable function

$? \equiv \mu$ -*recursive functions*

Church's hypothesis

initial functions

\subset *primitive recursive functions*

\subset *computable total functions*

(= μ -recursive total function)

\subset *computable partial function*

(= μ -recursive (partial) function)

(= partial recursive function)

1.3 Partial μ -Recursive Functions

computable partial functions

4th combinator of function: minimization(μ)

Define $f: \mathbb{N}^n \rightharpoonup \mathbb{N}$ from $g: \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ by

$$f(\bar{x}) = \mu y [g(\bar{x}, y) = 0]$$

$$= \text{minimum}(\mu) y \in \mathbb{N}, \text{ s.t. } g(\bar{x}, y) = 0$$

$$\wedge g(\bar{x}, z) \text{ is defined } 0 \leq \forall z < y.$$

Example: $g(x, y)$

$g: y \backslash x$	0	1	2	3	...
0	2	3	8	2	...
1	3	4	3	6	...
2	1	<u>0</u>	<u>X</u>	7	...
3	5	2	6	2	...
4	<u>0</u>	0	0	8	...
...					
f	4	2	X	?	...

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

$$f(x) = \mu y [\text{plus}(x, y) = 0]$$

$$f(x) = 0, \text{ if } x = 0$$

$$= \text{undefined}, x > 0.$$

$div: \mathbb{N}^2 \rightarrow \mathbb{N}$

$$div(x, y) = \mu z[(\sigma(x) \div (mult(z, y) + y)) = 0]$$

$$f(x) = \mu y[monus(x, y) = 0]$$

$$total\ function(f(x) = x)$$

minimization

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

$$f(\bar{x}) = \mu y[g(\bar{x}, y) = 0]$$

Assume the **partial** function g is **computable**,

for all $\bar{x} \in \mathbb{N}^n$ **do**

$$y = 0$$

while (not exit loop) **do**

if $g(\bar{x}, y) = 0$ **then** $f(\bar{x}) = y$; **exit loop** **fi**;

if $g(\bar{x}, y) = \text{undefined}$ **then**

$f(\bar{x}) = \text{undefined}$; **exit loop** **fi**;

$$y := \sigma(y)$$

od

od

$\therefore f$ is **computable and partial**,

if g is **computable and partial**.

μ -recursive function

*initial functions,
finite number of
combinations,
compositions,
primitive recursions, and
minimizations.*

$f \times g(\bar{x})$ is defined iff $f(\bar{x})$ and $g(\bar{x})$ are defined

$f \circ g(\bar{x})$ is defined iff $f(\bar{x})$ and $g(\bar{x})$ are defined

*$\mu y[g(\bar{x}, y)=0]$ is defined iff $g(\bar{x})$ and $h(\bar{x}, z, f(\bar{x}, z))$
is defined for $0 \leq \forall z < y$.*

initial functions

- \subset *primitive recursive functions*
- \subset μ -*recursive total functions*
(\cong *computable total functions*)
- \subset μ -*recursive partial functions*
(\cong *computable functions*)
- \subset *all functions*

Turing's thesis

*the class of Turing machine possesses
the computational power
of any computational system*

Church's thesis

*the class of μ -recursive (partial) function contains
all computational (partial) functions*

No one has proved it to be false

no one has found a partial function

that is computable but not partial recursive

***But! Turing's thesis and Church's thesis are one
and the same.***

TM = partial recursive function

Turing's thesis

TM = computation process

Church's thesis

μ RF = computation feature

TM = μ RF.

Turing Machine

A Turing machine is the sextuple of the form

$M = (Q, \Sigma, \Gamma, \delta, i, h)$ where

- (1) Q is a finite set of states,
 - (2) Σ is a finite set of input alphabets
 - (3) $\Gamma (\supseteq \Sigma)$ is a finite set of tape alphabets
 - (4) $\delta: (S - \{h\}) \times \Gamma \rightarrow S \times (\Gamma \cup \{L, R\})$
is the transition function
 - (5) $i \in S$ is the initial state
 - (6) $h \in S$ is the halting state
- $B \in \Gamma (B \notin \Sigma)$ is the blank symbol

Configuration: $\Gamma^* \times Q \times \Gamma^*$.

Let $p, q \in Q$; $X, Y \in \Gamma$; and $\alpha, \beta \in \Gamma^*$. Then
configuration (α, p, β)

state p , tape $\alpha\beta$ (scanning 1: β)

transition function

$$(1) \delta(p, X) = (q, Y) \quad p \xrightarrow{X/Y} q$$

$$(\alpha, p, X\beta) \Rightarrow (\alpha, q, Y\beta)$$

$$(2) \delta(p, X) = (q, L) \quad p \xrightarrow{X/L} q$$

$$(\alpha Y, p, X\beta) \Rightarrow (\alpha, q, YX\beta)$$

$$(3) \delta(p, X) = (q, R) \quad p \xrightarrow{X/R} q$$

$$(\alpha Y, p, X\beta) \Rightarrow (\alpha YX, q, \beta)$$

initial configuration for $w \in \Sigma^$*

$(\varepsilon, i, wBBB\dots) \quad w \in \Sigma^*$.

final configuration

$(\alpha, h, \beta) \quad \alpha, \beta \in \Gamma^*$.

abnormal termination

$(\varepsilon, p, X\beta) \rightarrow^{X/L} ?$

$L(M) = \{w \in \Sigma^* /$

$(\varepsilon, i, wBBB\dots) \Rightarrow^* (\alpha, h, \beta), \alpha, \beta \in \Gamma^*\}$

Basic Building Blocks

machine R , L , and x

$\forall X \in \Gamma, \delta_R(i, X) = (h, R)$ move one cell right

$\forall X \in \Gamma, \delta_L(i, X) = (h, L)$ move one cell left

$\forall Y \in \Gamma, \delta_X(i, Y) = (h, X)$ write x

combining the machines

$\rightarrow R \rightarrow X \rightarrow L$ or $\rightarrow RXL$

$\forall Y \in \Gamma, (\alpha, i, Y\beta) \Rightarrow^* (\alpha, h, X\beta)$

searching symbols

R_X : search for x to right of the initial position.

$(\alpha, i, \beta XZ) \Rightarrow^* (\alpha\beta, h, XZ)$ $\beta \in (\Gamma - \{X\})^*$.

$R_{\neg X}$: search for other symbols than X to right of

$(\alpha, i, \beta YZ) \Rightarrow^* (\alpha\beta, h, YZ)$ $\beta \in (\Gamma - \{X\})^*, Y \neq X$

$L_X, L_{\neg X}$: search left

abnormal termination

shift operations

S_R : shift one cell right

$(\alpha X, i, Y\beta) \Rightarrow^* (\alpha, h, X\beta)$ $X \neq B$

$(\alpha B, i, Y\beta) \Rightarrow^* (\alpha B, h, B\beta)$

to avoid abnormal termination

S_L : shift one cell left

$(\alpha X, i, Y\beta) \Rightarrow^* (\alpha X, h, \beta)$

Turing Computable Functions

Consider a partial function $f: \mathbf{N}^m \rightharpoonup \mathbf{N}^n$.

$$\begin{aligned} f(x_1, \dots, x_m) &= (y_1, \dots, y_n) \\ &= \text{undefined} \end{aligned}$$

input tape $(B x_1 B x_2 \dots B x_m B \dots)$

output tape $(B y_1 B y_2 \dots B y_n B \dots)$

A Turing machine $M = (S, \{0, 1, \#\}, \Gamma, \delta, i, h)$

initial configuration

$$(\varepsilon, i, w_1 \# w_2 \dots \# w_m B B \dots)$$

final configuration

$$(\varepsilon, h, v_1 \# v_2 \dots \# v_n B B \dots)$$

abnormal termination

w_i and v_j are binary representation
of x_i and y_j , respectively

The turing machine M **computes**
the μ -recursive partial function f

Turing Compatibility of μ -recursive Functions

Integer arguments

$$\Gamma = \{0, 1, B\},$$

$$f: \mathbf{N}^n \rightarrow \mathbf{N}^m.$$

Theorem 4.2 *Every μ -recursive function is Turing-computable.*

Proof

Initial functions are Turing-computable

ζ (zero)

σ (successor)

π_i^j (projection)

Partial functions constructed from Turing-computable partial functions using combination, composition, partial recursion and minimization are also Turing-computable.

μ -Recursive Nature of Turing Machines

computation power of Turing machine

restricted to the ability

to compute the μ -recursive functions

Let $|\Gamma| = b$. Then the content of the tape can be interpreted as a nonnegative integer of base b , written in reverse order.

Turing machine

partial function from \mathbf{N} to \mathbf{N} .

the partial function computed by

Turing machine is μ -recursive.

Theorem 4.3

Any computational process performed

by a Turing machine is actually the process of computing a μ -recursive function.

Proof

Let $M = (S, \Sigma, \Gamma, \delta, i, h)$ be a Turing machine

$f: \mathbf{N} \rightarrow \mathbf{N}$ be the μ -function computed by M

by interpreting the contents of the tape

as base $b = |\Gamma|$ integer representations

*in **reversed** order.(right blanks)*

state $|Q| = k$
 final state 0
 initial state 1
 other states 2, ..., $k-1$

transition

$mov(p, X) =$ 2 head moves right
 1 head moves left
 0 otherwise.

$sym(p, X) =$ Y write symbol Y
 X otherwise.

$state(p, X) =$ $q \in \{0, \dots, k-1\}$ state moves to q
 k if $p = 0$ or (p, X) is not valid.

mov , sym and $state$ are **primitive recursive**.

Configuration tuple (α, p, β)

New configuration tuple (w, p, n)

w tape content

integer of base $|\Gamma|$ in reverse order
 (right blanks)

p state number $\{0, \dots, k-1\}$

n head position $|\alpha\beta| = m, 1 \leq n \leq m.$

cursym: $\mathbb{N}^3 \rightarrow \mathbb{N}$

$$\text{cursym}(w, p, n) = \text{quo}(w, b^{n-1}) \div \text{mult}(b, \text{quo}(w, b^n))$$

nexthead, *nextstate*, *nexttape*: $\mathbb{N}^3 \rightarrow \mathbb{N}$

using *mov*, *sym*, *state*, and *cursym*

nexthead, *nextstate*, and *nexttape* are

primitive recursive

step: $\mathbb{N}^3 \rightarrow \mathbb{N}^3$

step = *nexthead* \times *nextstate* \times *nexttape*

step is ***primitive recursive***

run: $\mathbb{N}^4 \rightarrow \mathbb{N}^3$

$$\text{run}(w, p, n, 0) = (w, p, n)$$

$$\text{run}(w, p, n, t+1) = \text{step}(\text{run}(w, p, n, t))$$

run is ***primitive recursive***

stoptime: $\mathbb{N} \rightarrow \mathbb{N}$: number of steps to final states

$$\text{stoptime}(w) = \mu t [\pi_2^3(\text{run}(w, 1, 1, t)) = 0]$$

stoptime is **μ -recursive partial**

f: $\mathbb{N} \rightarrow \mathbb{N}$ (*nexttape*): final tape contents
the partial function computed by *M*

$$f(w) = \pi_1^3(\text{run}(w, 1, 1, \text{stoptime}(w)))$$

f is **μ -recursive partial**

Church-Turing thesis

operation approach(Turing's thesis)

Turing machine

functional approach(Church's thesis)

μ -recursive function

reenforce the confidence of our conjecture

Post system

Def. A Post system Π is defined by

$\Pi = (C, V, A, P)$ where

1) C is a finite set of **constants** with

$$C = C_N \cup C_T \text{ and } C_N \cap C_T = \emptyset,$$

2) V is a finite set of **variables**,

3) A is a finite set from C^* , called **axiom**, and

3) P is a finite set of **productions** of the form

$$x_1 V_1 x_2 \dots x_n V_n x_{n+1} \rightarrow y_1 W_1 y_2 \dots y_m W_m y_{m+1},$$

where $x_i, y_i \in C^*$, $V_i, W_i \in V$,

$$V_i \neq V_j \text{ for } i \neq j \text{ and } \cup_{i=1}^m W_i \subseteq \cup_{i=1}^n V_i.$$

any variable can appear at most once on the left
each variable on the right must appear on the left

In other words,

$$\{V_1, \dots, V_n\} \supseteq \{W_1, \dots, W_m\} \text{ and}$$

$$|V_1, \dots, V_n| = n \geq |W_1, \dots, W_m|.$$

$$\therefore f: \{1, \dots, m\} \rightarrow \{1, \dots, n\} = \{1, \dots, n\}^{\{1, \dots, m\}}.$$

$$x_1 V_1 x_2 \dots x_n V_n x_{n+1} \rightarrow y_1 V_{f(1)} y_2 \dots y_m V_{f(m)} y_{m+1} \in P$$

$$x_1 W_1 x_2 \dots x_n W_n x_{n+1} \Rightarrow y_1 W_{f(1)} y_2 \dots y_m W_{f(m)} y_{m+1}$$

$$L(\Pi) = \{w \in C_T^* \mid w_0 \Rightarrow^* w \text{ for some } w_0 \in A\}$$

$$\text{Ex. } C_T = \{a, b\}$$

$$C_N = \{\}$$

$$V = \{V_1\}$$

$$A = \{\varepsilon\}$$

$$P = \{V_1 \rightarrow aV_1b\}$$

$$\varepsilon \Rightarrow ab \Rightarrow aabb \Rightarrow \dots$$

$$\text{where } V_1 = \varepsilon.$$

Ex.

$$C_T = \{1, +, =\}$$

$$C_N = \{\}$$

$$V = \{V_1, V_2, V_3\}$$

$$A = \{1 + 1 = 11\}$$

$$P = \{V_1 + V_2 = V_3 \rightarrow V_11 + V_2 = V_31 \\ V_1 + V_2 = V_3 \rightarrow V_1 + V_21 = V_31\}$$

$$1 + 1 = 11 \Rightarrow 11 + 1 = 111 \\ \Rightarrow 11 + 11 = 1111$$

we can interpret the derivation

$$1 + 1 = 2 \Rightarrow 2 + 1 = 3$$

$$\Rightarrow 2 + 2 = 4$$

...

Theorem 13.6

*A language is **recursively enumerable** if and only if there exists some Post system that generates it.*

Proof

The derivation of Post system is completely mechanical, it can be carried out on a Turing machine.

For converse,

consider a unrestricted grammar $G = (N, T, P, S)$

$\Pi = (C, V_{\Pi}, A, P_{\Pi})$

$C_N = N, C_T = T, A = \{S\}, V_{\Pi} = \{V_1, V_2\}$, and

$P_{\Pi} = \{V_1\alpha V_2 \rightarrow V_1\beta V_2 / \alpha \rightarrow \beta \in P\}$

Rewriting System

Post system

*rewriting of strings in C^**

Turing machine

rewiring of strings in $Q \times C^ \times \Gamma^*$ (configuration)*

Matrix grammar

$$P = P_1 \cup P_2 \cup \dots \cup P_n$$

where $P_i = \{\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \dots\}$

ordered set

Whenever the first production of some P_i is applied, the second one must be applied in next.

Ex.

$$P_1: S \rightarrow S_1 S_2$$

$$P_2: S_1 \rightarrow aS_1, S_2 \rightarrow bS_2c$$

$$P_2: S_1 \rightarrow \varepsilon, S_2 \rightarrow \varepsilon$$

$$S \Rightarrow S_1 S_2 \Rightarrow aS_1 S_2 \Rightarrow aS_1 bS_2 c \Rightarrow aaS_1 bbS_2 cc \Rightarrow aab- \\ bcc$$

$$L = \{a^n b^n c^n \mid n \geq 0\}$$

Def. A language L is in $DTIME(T(n))$
 A language L is in $NTIME(T(n))$

$DTIME(T(n)) \subseteq NTIME(T(n))$

If $T_1(n) = O(T_2(n))$, $DTIME(T_1(n)) \subseteq DTIME(T_2(n))$

Thm. $DTIME(n^k) \subset DTIME(n^{k+1})$ for $k \geq 1$.

$L_{REG} \subseteq DTIME(n)$

$L_{CF} \subseteq DTIME(n^3)$

$L_{CF} \subseteq NTIME(n)$

L_{CS} every sentence of length n can be parsed in
 n^M where M depends on the grammar

But we can not say $L_{CS} \subseteq DTIME(n^M)$,

since the upper bound M is not known.

$L_{RE} \subseteq DTIME(f(n))$

There does not exist any $f(n)$.

The connection between Chomsky's hierarchy and the complexity is tenuous and not very clear.

Def. $P = \cup_{i \geq 1} DTIME(n^i)$
 $NP = \cup_{i \geq 1} NTIME(n^i)$

$P \subseteq NP$

*But it is not known if this containment is **proper**.*

Cook-Karp thesis

*A problem that is in P is called **tractable**,
and one that is not is called **intractable**.*

*But $2^{0.1n}$ is **intractable** whereas n^{100} is **tractable**.*

*An empirical observation that most practical
problems in P are in $DTIME(n)$, $DTIME(n^2)$,
or $DTIME(n^3)$.*

*An interesting problem whether or not
 $P = NP$.*

NP-complete problem

*that is as hard as any **NP** problem, and
in some sense is equivalent to all of them.*

Def. A language $L_1(\Sigma_1)$ is **polynomial-time reducible** to some language $L_2(\Sigma_2)$ if there exist a **deterministic Turing machine** by which any $w_1 \in \Sigma_1^*$ can be transformed to $w_2 \in \Sigma_2^*$ in a such way that $w_1 \in L_1$ if and only if $w_2 \in L_2$.

If L_1 is polynomial-time reducible to L_2 , and if $L_2 \in \mathbf{P}$ then $L_1 \in \mathbf{P}$. Similarly if $L_2 \in \mathbf{NP}$ then $L_1 \in \mathbf{NP}$.

Def. A language L is said to be **NP-complete** if $L \in \mathbf{NP}$ and if every $L' \in \mathbf{NP}$ is polynomial-time reducible to L .

If some L_1 is **NP-complete** and polynomial-time reducible to L_2 , then L_2 is also **NP-complete**.

If we can find a **deterministic polynomial-time algorithm** for any **NP-complete** language, then every language in **NP** is also in **P**, that is
NP = P.

Boolean expression*boolean constant 0(false), 1(true)**boolean variables**boolean operator \vee (or), \wedge (and), \neg (not)***conjunctive normal form***with variables x_1, x_2, \dots, x_n .* *$e = t_1 \wedge t_2 \wedge \dots \wedge t_m$ where the terms t_i are**where $t_i = s_1 \vee s_2 \vee \dots \vee s_p$* *where s_j is either x_k or $\neg x_k$.***Satisfiability Problem**

Given an expression e in conjunctive normal form, is there an assignment of values to the variables x_1, x_2, \dots, x_n that will make the value e true.

$$e_1 = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3)$$

satisfiable, $x_1=0, x_2=1, x_3=1$

$$e_1 = (x_1 \vee x_2) \wedge \neg x_1 \wedge \neg x_2$$

*not satisfiable***deterministic algorithm***exhaustive search: 2^n cases.***nondeterministic algorithm** *$O(n)$.*

The satisfiability problem

language problem

encode specific instance as a string that is

accepted iff the expression is satisfiable.

*this problem is **NP**-complete.*

Cook's theorem

*A large number of **NP**-complete has found.*

For all of them we can find exponential algorithm

But no one has discovered a polynomial-time alg.

∴ We believe that probably

$P \neq NP$ ($P \subset NP$)

But no one has produced an actual language

*in **NP** that is not in **P** or alternatively,*

no one has proven that no such language exists.

Open problem!