THEORY OF COMPUTATION
Formal Languages, Automata, and Complexity

Chap. 4 ***Computability***

J. Glenn Brookshear
The Benjamin/Cummings Pub. Comp., Inc.
1989

Chap. 4 Computability

Computation

operational *approach*

how *a computation is performed*

functional *approach*

what *a computation accomplishes*

Operational *approach*

define ***algorithms*** *rather than*

some ***functions*** *in a particular system*

Functional approach

define ***functions*** *computed rather than*

*means of computing(****algorithms****) them*

4.1 Foundation of Recursive Function Theory

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 $\rightarrow$ ***nonnegative integers***

tuples of nonnegative integers

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

$f(\bar{x}) \in \mathbf{N}^n \cup \{\varnothing\}$

Ex. *div:* $\mathbf{N}^2 \rightharpoonup \mathbf{N}$ *where* $div(x, y) = x/y$, *if* $y \neq 0$.

partial functions *includes*

strictly partial functions *and*

total functions

Primitive Recursive Functions

Three Initial Functions

1. ***zero function***, $\zeta: \mathbf{N}^0 \rightarrow \{0\}$ *from empty tuple to {0}*

$\zeta( ) = 0$

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

$\sigma(x) = x + 1$ *or ++x in C*

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

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

Ex.) $\pi_2^3(7, 6, 4) = 6$, $\pi_1^2(5, 17) = 5$, $\pi_2^2(5, 17) = 17$, $\pi_0^2(5, 17) = ( )$.

all of three ***initial*** *functions are* ***computable***.

Three combinators

1. ***Combination*** *of functions:* $\times$

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

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

$f \times g\,(\bar{x}) = (f(\bar{x}), g(\bar{x}))$ *where* $\bar{x} \in \mathbf{N}^k$.

Ex.) $\pi_1^3 \times \pi_3^3(4, 6, 8) = (4, 8)$

2. ***Composition*** *of functions:* $\circ$

Define $g \circ f: \mathbf{N}^k \to \mathbf{N}^n$ *from*

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

$g \circ f\,(\bar{x}) = g(f(\bar{x}))$ *where* $\bar{x} \in \mathbf{N}^k$.

Ex.): $\sigma \circ \zeta(\,) = \sigma(0) = 1$.

3. ***Primitive recursion*** *of functions: f*

Example*: f:* $N^2 \rightarrow N$ *such that f(x, y) is the number of nodes in a* ***full*** *x-nary* ***balanced*** *tree whose depth is y.*

Recursive thinking*!*

(1) level n has x^n *nodes*

(2) depth y+1: add $x^{y+1} = x^y \cdot x$ *nodes to f(x, y):depth y.*

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

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

f is ***computable***

$\underline{f(3, 2)} =_{(R)} \underline{f(3, 1) + 3^{\underline{1}} \cdot 3}$

$= \underline{f(3, 1)} + 9 =_{(R)} \underline{f(3, 0) + 3^{\underline{0}} \cdot 3} + 9$

$= \underline{f(3, 0)} + 12 =_{(B)} \underline{3}^{\underline{0}} + 12 = 13.$

In general,

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

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

Define $f: \mathbf{N}^{k+1} \to \mathbf{N}^m$ from

$g: \mathbf{N}^k \to \mathbf{N}^m$ and $h: \mathbf{N}^{k+m+1} \to \mathbf{N}^m$ by $^\forall x \in \mathbf{N}^k$

$f(\bar{x}, 0) =_B g(\bar{x})$ or $f(\bar{x}, \zeta(y)) = g(\bar{x})$

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

where $\bar{x} \in \mathbf{N}^k$ and $y \in \mathbf{N}$.

$f(\bar{x}, 3) =_R h(\bar{x}, 2, f(\bar{x}, 2)) =_R h(\bar{x}, 2, h(\bar{x}, 1, f(\bar{x}, 1)))$

$=_R h(\bar{x}, 2, h(\bar{x}, 1, h(\bar{x}, 0, f(\bar{x}, 0)))) =_B h(\bar{x}, 2, h(\bar{x}, 1, h(\bar{x}, 0, \bar{x}, 0, g(\bar{x}))))$.

$plus: \mathbf{N}^2 \to \mathbf{N}$

$plus(x, 0) =_B \pi_1^1(x)$

$plus(x, y+1) =_R \sigma \circ \pi_3^3(x, y, plus(x, y)) = \sigma \circ plus(x, y)$

Ex.) $plus(100, 2) =_R \sigma \circ \pi_3^3(x, y, plus(100, 1)) = \sigma \circ plus(100, 1)$

$=_R \sigma \circ \sigma \circ plus(100, 0) =_B \sigma \circ \sigma \circ (100) = \sigma \circ 101 = 102.$

Primitive recursive functions

constructed from ***initial*** *functions*($\zeta(\,)$, $\sigma(x)$, $\pi_n^m(\bar{x})$)

with ***finite number*** *of* ***combinations***($\times$), ***compositions***($\circ$),

and ***primitive recursions***.

Every ***primitive recursive function*** *is* ***total*** *and* ***computable***, *since*

initial *functions are* ***total*** *and* ***computable***.

finite number of $\times$, $\circ$, *and* ***primitive recursions***

applied is ***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 *functions* $\subset$ ***computable total*** *functions*
$\subset$ ***computable*** *(****partial****) functions*

4.2 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***

Examples of primitive recursive functions

Constant functions

K_m^n: $\boldsymbol{N}^n \rightarrow \{m\}$, $m, n \in \boldsymbol{N}$.

where $K_m^n(\bar{x}) = m$, $\bar{x} \in \boldsymbol{N}^n$.

For $m = 0$,

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

$K_m^0 = \zeta \circ \sigma \circ \ldots \circ \sigma = \zeta \circ \sigma^m = m$

$\therefore K_m^0$ *is* ***primitive recursive***

For $n > 0$,

$K_m^n(\bar{x}, 0) =_B K_m^{n-1}(\bar{x})$

$K_m^n(\bar{x}, \sigma(y)) =_R \pi_{n+2}^{n+2}(\bar{x}, y, K_m^n(\bar{x}, y)) = K_m^n(\bar{x}, y)$

$\therefore K_m^n$ *is* ***primitive recursive***$(n \geq 0)$.

$K_m^n(k_1, \ldots, k_n) =_R K_m^n(k_1, \ldots, k_n\text{-}1) =_R \ldots$

$=_R K_m^n(k_1, \ldots, k_{n-1}, 0) =_B K_m^{n-1}(k_1, \ldots, k_{n-1}) = \ldots$

$=_B K_m^1(k_1) =_R \ldots =_R K_m^1(0) =_B K_m^0() = \sigma \circ \ldots \circ \sigma \circ \zeta\,() = \ldots = m.$

Constants are also primitive recursive functions

$7 = K_7^5$

$(2, 5) = K_2^7 \times K_5^3$

Arithmetic functions *from* ***constant*** *and plus*

$mult(x, 0) = K_0^1(x)$

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

$mult(x, 0) = 0$

$mult(x, y+1) = plus(x, mult(x, y))$

$expo(x, 0) = K_1^1(x)$

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

$expo(x, 0) = 1$

$expo(x, y+1) = mult(x, expo(x, y))$

predecessor function

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

$pred(0) = \zeta() = 0$

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

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

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

$monus(x, 0) = x$

$monus(x, y+1) = pred(monus(x, y))$

$monus \equiv \dot{-}$ $\quad x \dot{-} y = x - y$, *if* $x \geq y$; 0, *otherwise.*

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

eq(x, y) = 1 if $x = y$, *0 if* $x \neq y$

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

$=monus(1, plus(monus(x, y), monus(y, x)))$

Example:

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

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

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

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

gt. lt, ge. le.: $\mathbf{N}^2 \rightarrow \mathbf{N}$

left for exercise

Beyond Primitive Recursive Functions

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)

primitive recursive functions $\overset{?}{\subset}$ *computable total functions*

Ackermann function*(W. Ackermann in 1928)*

$A: \mathbf{N}^2 \to \mathbf{N}$

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

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

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

Ackermann function is ***computable total****(App. B)*

but ***not primitive recursive****.*

$A(3, 2) = A(2, A(3, 1))$

$= A(2, A(2, A(3, 0)))$

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

$= A(2, A(2, A(1, A(2, 0))))$

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

$= A(2, A(2, A(1, A(0, A(1, 0)))))$

$= A(2, A(2, A(1, A(0, A(0, 1)))))$

$= A(2, A(2, A(1, A(0, \underline{2}))))$

$= \ldots$

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

Proof

the number of p.r. functions is ***countable***

(finite combination, composition, p. rec.)

$\therefore f_1, \ldots, f_n, \ldots$

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

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

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

($\therefore$ *no. of computable functions is* ***uncountable****)*

computable function

$^{?}\equiv$ ***μ-recursive functions***

Church's hypothesis

4.3 Partial μ-Recursive Functions

computable partial functions

***4th combinator** of function: **minimization**(μ)*

Define f: $\mathbf{N}^n \to \mathbf{N}$ from g: $\mathbf{N}^{n+1} \to \mathbf{N}$ by

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

$= minimum(\mu)\ y \in \mathbf{N}$, .э. $g(\bar{x}, y) = 0$

$\wedge\ g(\bar{x}, z)$ *is defined* $0 \le {}^{\forall} z < y$.

Example: g(x, y)

g: *y\x*	*0*	*1*	*2*	*3*	…
0	*2*	*3*	*8*	*2*	…
1	*3*	*4*	*3*	*6*	…
2	*1*	***0***	***X***	*7*	…
3	*5*	*2*	*6*	*2*	…
4	***0***	*0*	***0***	*8*	…
…					
f	*4*	*2*	*X*	*?*	…

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

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

$f(x) = 0$, *if* $x = 0$

$= undefined, x > 0.$

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

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

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

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

minimization

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

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

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

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

y = 0

while *(not exit loop)* ***do***

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

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

$f(\bar{x}) =$ *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.

$\mu y[g(\bar{x}, y)=0]$ *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 *μRF = computation **feature***

TM = μRF.

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) $\delta: (Q - \{h\}) \times \Gamma \rightarrow Q \times (\Gamma \cup \{L, R\})$

is the transition function

(5) $i \in Q$ *is the initial state*

(6) $h \in Q$ *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*)*

*three **normalized** transition function*

(1) $\delta(p, X) = (q, Y)$ $\quad p \rightarrow^{X/Y} q$ $\quad$ ***replace** X to Y*

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

(2) $\delta(p, X) = (q, L)$ $\quad p \rightarrow^{X/\leftarrow} q$ $\quad$ *move **left***

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

(3) $\delta(p, X) = (q, R)$ $\quad p \rightarrow^{X/\rightarrow} q$ $\quad$ *move **right***

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

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

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

final configuration

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

abnormal termination

$(\varepsilon, p, X\beta) \rightarrow^X$ *?*

$L(M) = \{w \in \Sigma^* \mid (\varepsilon, i, wBBB\ldots) \Rightarrow^* (\alpha, h, \beta), \alpha, \beta \in \Gamma^*\}$

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.*

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

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

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

$L_X, L_{\neg X}$*: search for ... left to ... abnormal termination*

Shift operations

S_R: *shift one cell right*

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

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

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

$f(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$

$= undefined$

input tape $(B\, x_1\, B\, x_2 \ldots B\, x_m\, B \ldots)$

output tape $(B\, y_1\, B\, y_2 \ldots B\, y_n\, B \ldots)$

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

initial configuration

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

final configuration

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

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

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

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

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

Proof

1. Initial functions are Turing-computable

ζ*(zero)*

σ*(successor)*

π_i^j*(projection)*

2. 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 ***N*** *to* ***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 $M = (Q, \Sigma, \Gamma, \delta, i, h)$ be a Turing machine
$f: \boldsymbol{N} \rightarrow \boldsymbol{N}$ be the μ-recursive 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) = $q \in \{0, \ldots, k\text{-}1\}$ *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* $\{0, \ldots, k\text{-}1)\}$

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

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

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

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

using mov, sym , state, and cursym

nexthead, nextstate, and nexttape are

primitive recursive

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

$step = nexthead \times nextstate \times nexttape$

step is ***primitive recursive***

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

$run(w, p, n, 0) = (w, p, n)$

$run(w, p, n, t+1) = step(run(w, p, n, t))$

run is ***primitive recursive***

*stoptime: **N** → **N**: number of steps to final states*

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

stoptime is μ***-recursive partial***

*f: **N** → **N** (nexttape): final tape contents*

the partial function computed by M

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

f is μ***-recursive partial***

Church-Turing thesis

operation approach(Turing's thesis)

Turing machine

functional approach(Church's thesis)

μ*-recursive* ***partial*** *function*

To reenforce the confidence of our conjecture, in other text ...

Turing-Church's thesis

Theorem 4.2 *Every μ-recursive function is Turing-computable.*
Theorem 4.3
Any computational process performed by a Turing machine is actually the process of computing a μ-recursive function.

$\therefore$ *TM* = μ*RF*.
Any ***computation*** *is TM or μ-recursive function.*

What further?
not yet*!*

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* $C = C_N \cup C_T$ *and* $C_N \cap C_T = \varnothing$,

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

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

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

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

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

$V_i \neq V_j$ *for* $i \neq j$ *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,

$$\{V_1, \ldots, V_n\} \supseteq \{W_1, \ldots, W_m\} \text{ and } |V_1, \ldots, V_n| = n \geq |W_1, \ldots, W_m|.$$

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

$$x_1 V_1 x_2 \ldots x_n V_n x_{n+1} \rightarrow y_1 V_{f(1)} y_2 \ldots y_m V_{f(m)} y_{m+1} \in P$$
$$x_1 w_1 x_2 \ldots x_n w_n x_{n+1} \Rightarrow y_1 w_{f(1)} y_2 \ldots y_m w_{f(m)} y_{m+1}$$

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

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

$C_N = \{\}$

$V = \{V_1\}$

$A = \{\varepsilon\}$

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

$\varepsilon \Rightarrow ab \Rightarrow aabb \Rightarrow \ldots$

where $V_1 = \varepsilon$.

Ex.

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

$C_N = \{\}$

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

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

$P = \{V_1 + V_2 = V_3 \rightarrow V_1 1 + V_2 = V_3 1$

$V_1 + V_2 = V_3 \rightarrow V_1 + V_2 1 = V_3 1\}$

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

$\Rightarrow 11 + 11 = 1111$

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

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

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

ordered set

Whenever the first production of some P_i *is applied,*

the second one must be applied in next.

Ex. $P_1: S \rightarrow S_1S_2$

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

$P_3: S_1 \rightarrow \varepsilon, S_2 \rightarrow \varepsilon$

$S \Rightarrow S_1S_2 \Rightarrow aS_1S_2 \Rightarrow aS_1bS_2c \Rightarrow aaS_1bbS_2cc \Rightarrow aabbcc$

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

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 exists any f(n).

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

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

$\boldsymbol{NP} = \cup_{i \geq 1} NTIME(n^i)$

$\boldsymbol{P} \subseteq \boldsymbol{NP}$

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 $w_1 \in$*

Σ_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 $L_2 \in$ **P** then $L_1 \in$ **P**. Similarly if $L_2 \in$ **NP** then $L_1 \in$ **NP**.*

***Def**. A language L is said to be **NP-complete** if $L \in$ **NP** and if every L' $\in$ **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 $x_1, x_2, \ldots, x_n$.

$e = t_1 \wedge t_2 \wedge \ldots \wedge t_m$ where the terms t_i are

where $t_i = s_1 \vee s_2 \vee \ldots \vee s_p$

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 $x_1, x_2, \ldots, x_n$ 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.

$\therefore$ *We believe that probably*

$\boldsymbol{P \neq NP}$ $\boldsymbol{(P \subset NP)}$

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!