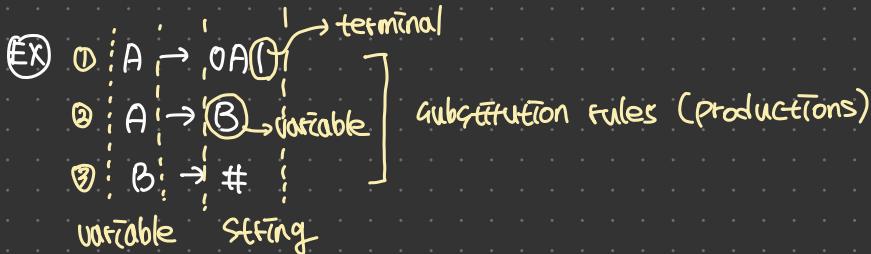


2. Context Free Languages

- ⇒ NFA, DFA 보다 more powerful (human languages)
- ⇒ recursive structure
- ⇒ 구분자 → grammar → language

2.1 Context Free Grammars



①의 A : start variable

① derivation : the sequence of substitutions to obtain a string

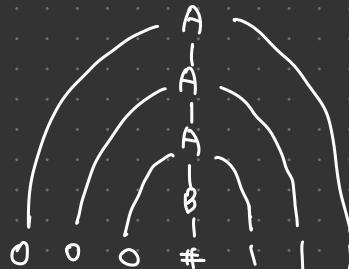
↳ variable로 input에서 시작할 때까지 substitution rule을 적용
⇒ 하나의 과정이 derivation.

Ex. 000#||| derivation

$A \Rightarrow \alpha A \beta \Rightarrow \underset{①}{\alpha} \underset{②}{A} \underset{③}{\beta} \Rightarrow \underset{①}{000} \underset{②}{A} \underset{③}{|||} \Rightarrow \underset{①}{000} \underset{②}{B} \underset{③}{|||} \Rightarrow \underset{①}{000} \underset{②}{\#} \underset{③}{|||}$

② parse tree : derivation을 tree처럼 표현

Ex. 000#|||



③ CFL (context-free language)

: context-free grammar에 의해 생성될 수 있는 language

⇒ grammar을 표기할 때 편의를 위해 "I", "A" 사용

ex. $A \rightarrow 0A1 \mid B$

④ Formal Definition of a CFG

Definition 2.2

A **context-free grammar** is 4-tuple (V, Σ, R, S) where

- ① V is a finite set of **variables**,
- ② Σ is a finite set, disjoint from V , called the **terminals**,
- ③ R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, and
- ④ $S \in V$ is the **start variable**.

i) **yield** : $A \rightarrow w$ (rule) 을 적용하여

$uAv \Rightarrow uwv$ (derivation) 이 될 때
yield라고 표현

ii) **derive** : $U \Rightarrow U_1 \Rightarrow U_2 \dots \Rightarrow U_k \Rightarrow v$ ($k \geq 0$)

∴ U derives v

iii) **language of the grammar** : $\{w \in \Sigma^* \mid S \xrightarrow{*} w\}$
 \uparrow
terminal 포함.

ex) $\Sigma = \{a, b, c, \dots, z, \square\}$ → blank symbol

iv) Often we specify a grammar by writing down only its rules.
terminals 뒤에 대해서 derivation

④ G₄ = (V, Σ, R, ⟨EXPR⟩)

⟨EXPR⟩ → ⟨EXPR⟩ + ⟨TERM⟩ | ⟨TERM⟩

⟨TERM⟩ → ⟨TERM⟩ × ⟨FACTORY⟩ | ⟨FACTORY⟩

⟨FACTORY⟩ → (⟨EXPR⟩) | a

↳ parsing : program의 meaning 있는 parse tree

⑤ Designing CFG

i) the union of simpler CFGs.

: 각각의 grammars들은 simpler grammars의 rule들을 combining, adding으로 만들 수 있다.

ex) 90^n | n ≥ 0 } ∪ 9 | 0^n | n ≥ 0 }

① S → AIB

② A → 0A1 | ε

③ B → 1B0 | ε

ii) CFG가 R.L이면 DFA로 푸기 Design

a. variable R_i of CFG ⇒ state z_i of DFA

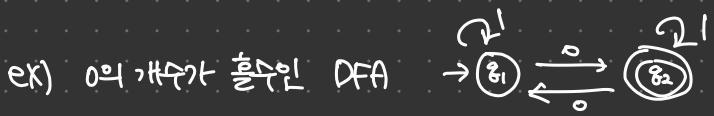
b. rule R_i → aR_j of CFG

⇒ transition f(z_i, a) = z_j of DFA

c. rule R_i → ε of CFG

⇒ z_i가 accept state of DFA

d. Root start variable of CFG ⇒ z₀는 start state



$$f(q_1, 0) = q_2 \rightarrow R_1 \rightarrow OR_2$$

$$f(q_1, 1) = q_1 \rightarrow R_1 \rightarrow IR_1$$

$$f(q_2, 0) = q_1 \rightarrow R_2 \rightarrow OR_1$$

$$f(q_2, 1) = q_2 \rightarrow R_2 \rightarrow IR_2$$

$$F = \{q_2\} \rightarrow R_2 \rightarrow \epsilon$$

$$\therefore \text{CFG} \Rightarrow R_1 \rightarrow OR_2 \mid IR_1 \\ R_2 \rightarrow OR_1 \mid IR_2 \mid \epsilon$$

- iii) 특정한 CFG들은 서로 "linked" 되어 있는 두 개의 substitution 포함
 → 유한한 정도의 variable을 기억해야 함
 → 각각 substitution들이 서로 correspond인지 확인

ex) $0^n 1^n \mid n \geq 0$

$R \rightarrow uRv$ 라는 rule은 실제해야 함.

→ 0의 개수가 match 되도록

- iv) 더 복잡한 CFL들은 recursive하게 특정한 구조를 포함

ex) $E \rightarrow E + T \mid T$

$T \rightarrow T \times F \mid F$

$F \rightarrow __ \mid a$

}

arithmetic expression

반복해서 0과 대체됨.

⑥ Ambiguity

: 같은 string이 다른 여러 가지 방법으로 생성될 수 있는 grammar

⇒ 여러 개의 parse tree를 가지고 이를 통해 다른 여러 meaning:
(programming language와 같은 language가 undesirable)

→ 허당 string : is derived ambiguously

허당 grammar : ambiguous 허다고 표현

⇒ derivation : 순서가 다르면 다른 derivation.

↳ derivation이 다르더라도 parse tree가 같으면
같은 grammar (그리는 순서만 다르면)

→ parse tree가 두개 이상이면 ambiguous

→ leftmost derivation : derivation의 순서를 가장 원쪽부터로 정해줌.

→ leftmost derivation이 두개 이상이면 ambiguous

∴ A string w is derived ambiguous in CFG G
if it two or more different leftmost derivations.

→ Grammar G is ambiguous if it generates some string
ambiguously

f) Inherently ambiguous : Some CFL은 ambiguous한 CFG에
의해서만 generate 가능

But,
어떻게 고려나에
여러 결과점

⑦ Chomsky Normal Form

Simplified form으로 만드는게 보통 편함

Definition 2.8

- A context-free grammar is in **Chomsky normal form** if every rule is of the form
 - $A \rightarrow BC$ variable 두개
 - $A \rightarrow a$ terminal 하나
- where a is any terminal and A, B , and C are any variables—except that B and C may not be the start variable.
- In addition we permit the rule $S \rightarrow \epsilon$ where S is the start variable.

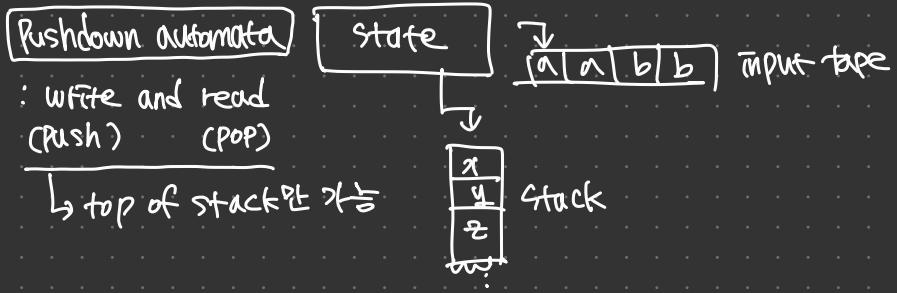
→ Every CFL은 CFG로의 편리한 표현 가능
(중령 쌩歹)

2.2 Pushdown Automata

: "Stack" 을 사용 \rightarrow control 할 수 있는 finite한 amount의 memory
 \rightarrow non-regular language를 recognize 가능하게 함

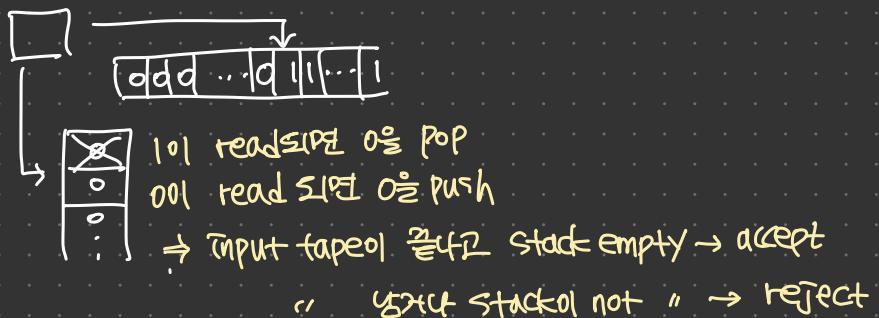
PDA와 CFG는 equivalent in power \Rightarrow useful !

⑥ schematic representation of a finite automaton



\therefore PDA : unlimited amount of information을 hold 가능

ex) $10^n 1^n \mid n \geq 0 \rightarrow$ no limit한데까지 가능



+ nondeterministic PDA의 power \supseteq deterministic PDA Power
 \hookrightarrow not equivalent

우리가 다룬 PDA는 NPPA

① Formal Definition of a PDA

: input alphabet Σ / stack alphabet $\Gamma \Rightarrow$ 같아도 됨

\rightarrow transition function이 중요함

$\text{① } f: Q \times \Sigma \times \Gamma \rightarrow$

\hookrightarrow input reading 없이 혹은 stack pop 없이

다음 state로 move 하는 것만 가능

$\text{② nondeterminism} \Rightarrow f: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)$

Definition 2.13 NPPA

A **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ , and F are all finite sets, and

- ① Q is the set of states, **finite**
- ② Σ is the input alphabet,
- ③ Γ is the stack alphabet,
- ④ $\delta: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)$ is the transition function,
- ⑤ $q_0 \in Q$ is the start state, and $\Rightarrow \epsilon$ 포함하면 accept state여도 됨
- ⑥ $F \subseteq Q$ is the set of accept states. \hookrightarrow state로 accept로 만들지 않아도 \in accept 가능

(Ex) $M = (Q, \Sigma, \Gamma, \delta, z_0, F)$

$w_i \in \Sigma^*$, $t_0, t_1, \dots, t_m \in Q$, $g_0, g_1, \dots, g_m \in \Gamma \epsilon^*$

$w = w_1 w_2 w_3 w_4 \dots w_n \rightarrow \text{input}$

$\Gamma = [t_0] \quad [t_1] \quad [t_2] \quad [t_3] \quad \dots \quad [t_m] \rightarrow \text{state}$

$S = [g_0] \quad [g_1] \quad [g_2] \quad [g_3] \quad \dots \quad [g_m] \rightarrow \text{stack}$

① start : $t_0 = z_0$, $g_0 = \epsilon$ \rightarrow empty stack

② For $i=0 \dots m-1$

$(\Gamma_{i+1}, b) \in \delta(t_i, w_{i+1}, a)$

\rightarrow stack이 있으면 풀어쓰기

③ $t_m \in F \rightarrow \text{accept}$

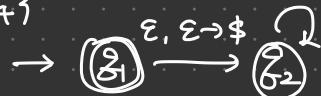
(Ex 2.14) $CFL \rightarrow \{0^n | n \geq 0\}$

$M_1 = (Q, \Sigma, \Gamma, \delta, z_0, F)$

$Q = \{z_1, z_2, z_3, z_4\}$

$0, \epsilon \rightarrow 0$

$\Sigma = \{0, 1\}$



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

$F = \{z_1, z_4\}$

δ	0	ϵ	1	ϵ	0	ϵ
z_1	$\{0, \$\}$	$\{\epsilon\}$	$\{0\}$	$\{\$\}$	$\{0, \$\}$	$\{\epsilon\}$
z_2	$\{(z_2, 0)\}$	$\{(z_3, \epsilon)\}$			$\{(z_3, \$)\}$	
z_3		$\{(z_3, 0)\}$	$\{(z_4, \epsilon)\}$		$\{(z_4, \$)\}$	
z_4						

A) : $(B, a) \rightarrow a, b \rightarrow c$ ⑤에서 ④로 가는 transition
: a를 입력한 stack을 pop한 후에 C로 연결

- Σ : ① $\Sigma \rightarrow \alpha$: α 를 push
 ② $\alpha \rightarrow \Sigma$: α 를 pop
 ③ $\Sigma \rightarrow \Sigma$: 아무일도 X

$$\Rightarrow w = 0011$$



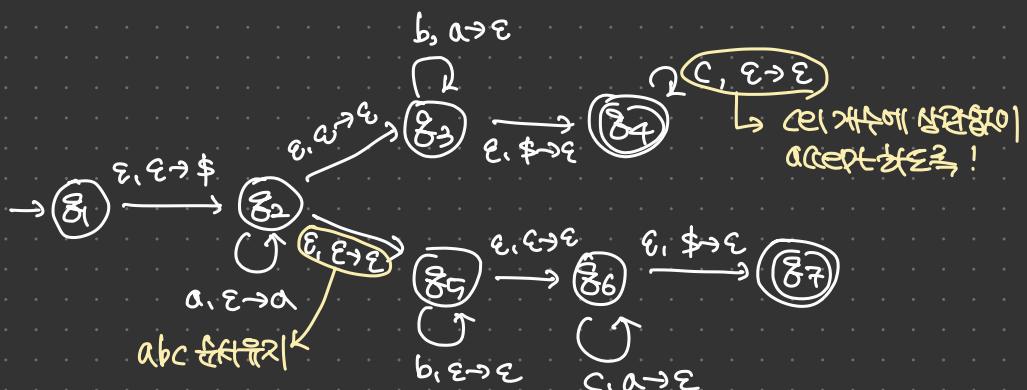
[Ex 2.16] $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i=j \text{ or } j=k\}$

↳ Nondeterminism 사용

(PDA는 pop한 것을 다시 stack에 넣을 수 X)

\Rightarrow branch은 2개로 두어서

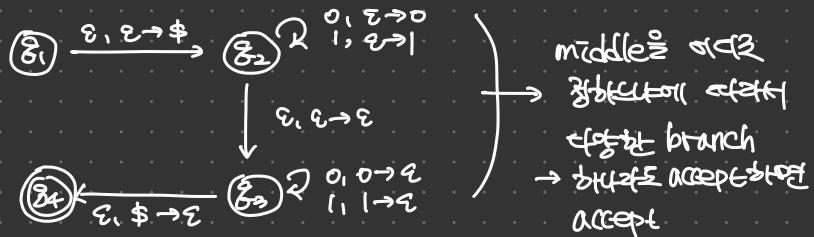
$a^i b^j = b^j a^i$ 같은 경우 $b^j a^i = C a^i b^j$ 형태로 맞는지 check



NPDA : 어떤 문자로 가도 빠져나오지 accept하면 accept

Ex 2.18 $\{ww^R \mid w \in \{0,1\}^*\}$

→ w의 reverse



② Equivalence with CFG

[Th 2.20] CFG \leftrightarrow PDA \Rightarrow 풀이 power가 같음

Pf. 1 : CFG \rightarrow PDA

\because CFL : A ↑ generate
 CFG : G
 PDA : P $\xrightarrow{\text{equivalent 증명!}}$

i) derivation : 중간단계를 만들어냄

ii) nondeterminism : 어떤 단계에서든 derivation들의

substitution인지 (중간단계) 찾아냄.

iii) substitution 처리 \rightarrow stack

iv) variable은 저장 X, substitution은 terminal!

Ex) ① $S \rightarrow 0S1$

② $S \rightarrow \epsilon$

i) informal description

① \$: the start variable on the stack

④ top of the stack이 variable ⑤이면,
nondeterministically ④가 rule 2의 뒷부분을 적용하여
getting 2가 됨

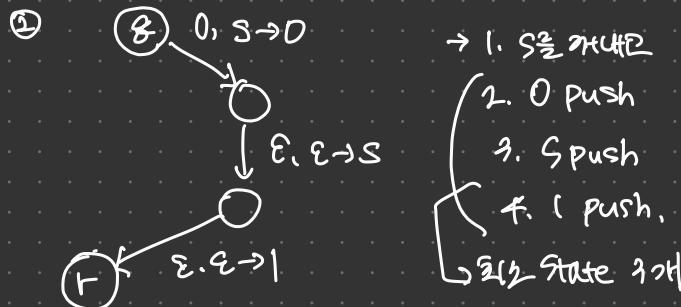
⑤ top of the stack이 terminal ⑥이면,
read the next symbol from the input
and compare it to ⑥
만약 두 symbol이 같지 않다면,
this branch will be reject

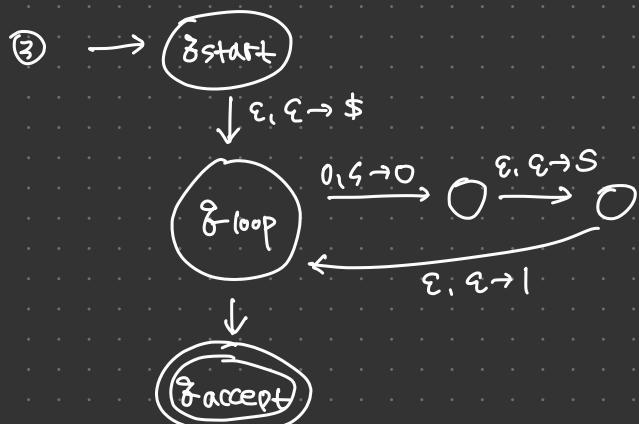
⑥ top of the stack이 \$ 이면, accept state!

ii) formal construction.

$$P = (Q, \Sigma, \Gamma, f, q_{start}, F)$$

→ 첫영문은 위해 초기 state를 넣었다가 원상복구



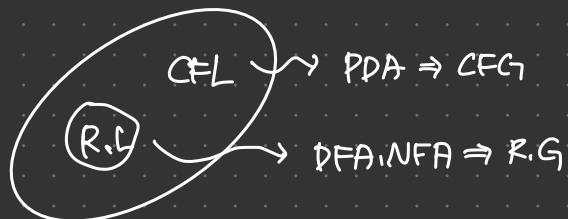


pf2 : PDA \rightarrow CFG

\because PDA : P \rightarrow P \neq acceptable strings generate
 $\text{CFG} : G$

\Rightarrow skip.

Corollary
2.32



2.3 Non-Context Free Languages

\rightarrow Context freest 아닌 것을 증명

\rightarrow Pumping lemma for CFL!

① Pumping lemma

A : CFL

p : pumping length

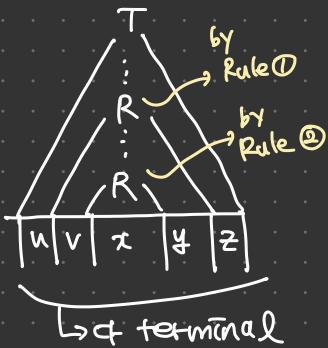
$$s \in A, |s| \geq p \rightarrow s = uvxyz$$

i) for each $i \geq 0$, $uv^i xy^i z \in A$ $\rightarrow \begin{cases} s = uvxyz \in A \\ s' = u v^2 x y^2 z \in A \end{cases}$

ii) $|vxy| > 0 \rightarrow v \neq \epsilon \text{ or } y \neq \epsilon$

~~iii) $|vxy| \leq p$~~ \rightarrow pumping length

PF Idea) Let. $A : \text{CFL} \leftarrow \text{generate}$
 $G : \text{CFG}$

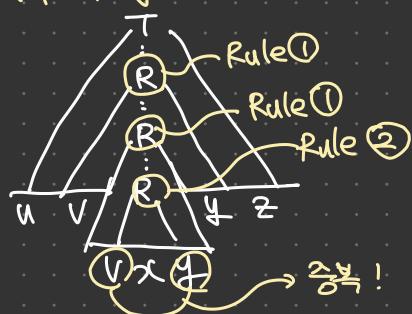


$s \in A$, s 가 충분히 긴 string이라고 가정

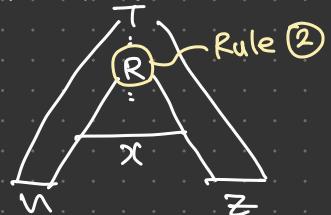
\Rightarrow parse tree도 매우 tall

- i) CFG의 경우, variable 개수 $<$ pass 개수
 \rightarrow 충복되는 variable 발생
- ii) 충복되는 variable 발생 \rightarrow 같은 Rule로 형성되는 것이 반복
 \rightarrow 충복되는 terminal 발생

#Pumping up



#Pumping down



PF) G : CFG

A : CFL

?) b : the maximum number of symbols

ex) $S \rightarrow OS | (길이: 3)$ $\rightarrow b = 3$
 $S \rightarrow \# \quad (\text{길이: } 1)$

parse tree : 각 node가 최대 b 개

$\rightarrow h$ step $\rightarrow b^h$ 개가 최대 node

\rightarrow parse tree 높이가 h \rightarrow 가장 긴 string은 b^h

\Rightarrow 만약 string이 b^{h+1} 길이면 parse tree는 $h+1$ high.

ii) $|V|$: the number of variables in G

iii) P , pumping length $= b^{|V|+1}$

(\because variable이 $(|V|+1)$ 개 이상 \rightarrow 무조건 증복 발생)

iv) $s \in A$, $|s| \geq P$ 일 때,

의 parse tree 높이 $\geq |V|+1$

$$\Rightarrow b^{|V|+1} > b^{|V|+1}$$

v) how to pump any such string s

T : parse tree를 중 하나

\hookrightarrow 이 중 가장 적은 node를 가진 parse tree 선택

T 의 height $> |V|+1$

T 의 Node $\uparrow > |V|+2$

\Rightarrow 하나의 terminal \rightarrow 최소 $(|V|+1)$ 개의 variable 필요

→ But, G 는 n 개의 variable을 가짐 \rightarrow 중복 필요

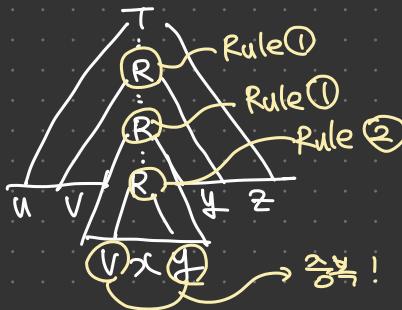
vi) R : 가장 아래에 나오는 반복하는 variable

\hookrightarrow generates v^{∞}

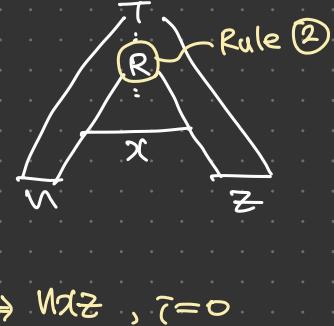
smaller sub tree

vii) Condition 1 : $uv^ix^jz \in A$, $i \geq 0$

#Pumping up

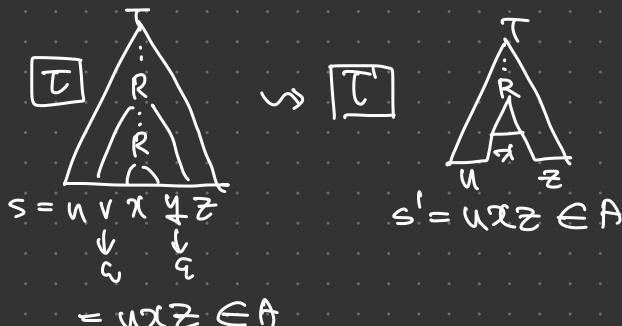


#Pumping down



$\rightarrow uv^ix^jz$, $i \geq 1$

viii) Condition 2 : $v = \epsilon$ and $y = \Sigma$



$= ux^{\infty}z \in A$

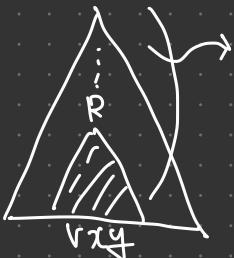
\rightarrow nodeel 가능성 : $T > T'$

But, T 은 가장 적은 node 가지는 parse tree

가장 짧았기에 impossible

vxi) Condition 3 : $|vxy| \leq p$

R : vxy를 generate하는 5번의 parse tree



R이 빙복 가능

\Rightarrow Variable은 $(vl+1)$ 개가 되는 순간 빙복

\rightarrow height : $(vl+1)$

node 개수 : $(vl+2)$

$\rightarrow |vxy| \leq b^{vl+1} = p$

② Pumping lemma example

Ex 2.36

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

① B가 CFL이라고 가정.

② By contradiction 증명

③ P가 pumping length

$$s = a^p b^p c^p, s \in B, |s| \geq p$$

$s = uvxyz$ 일 때 uv^nxy^z 증명

④ condition 2에 의해 $u \neq \emptyset$ and $y \neq \emptyset$

Case 1. v가 a, b, c 중 하나만 가지고 있다.

$$s = \underbrace{aa \dots a}_P \underbrace{bb \dots b}_P \underbrace{cc \dots c}_P$$

$$s' = a^{p+k} b^p c^p \notin B$$

Case2. yet \exists 가 a,b,C 중 두가지 이상

$$s = \underbrace{aa}_{\text{u}} \cdots \underbrace{abb}_{\text{v}} \cdots \underbrace{bcC}_{\text{w}} \cdots \underbrace{c}_{\text{z}}$$

$$s' = a \cdots abb \cdots ba \cdots ab \cdots bc \cdots c$$

$$\notin B$$

개수는 맞을 수 있지만 순서가 맞지 않음.

⑤ Case 1, Case 2 중 무조건 CFL이거나

B는 CFL이 아님.

Ex 2.17 $C = \{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$

① C가 CFL이라고 가정

② By Contradiction 증명

③ pumping length

$$s = a^p b^p c^p, s \in C, |s| \geq p$$

$s = uvxyz$ 일 때 $s = uxz \notin C$ 증명

Case 1. yet \exists 가 하나의 alphabet만 가지고 있는 경우

1. a : a가 없는 경우 \Rightarrow yet \exists 가 b 혹은 c에만

$$s = \underbrace{a \cdots a}_{\text{u}} \underbrace{b \cdots b}_{\text{v}} \underbrace{c \cdots c}_{\text{z}} \quad (\because |vxy| \leq p)$$

$$s' = \underbrace{a \cdots a}_{\text{u}} \underbrace{b \cdots b}_{\text{v}} \underbrace{c \cdots c}_{\text{z}} \notin C$$

P보다 작음

\Rightarrow c의 개수 < p

\Rightarrow b도 마찬가지

1.b : b가 없는 경우

① $s = a \cdots a b \cdots b c \cdots c$
 $\underbrace{a \cdots a}_{uvxy} \underbrace{b \cdots b}_{xz} \underbrace{c \cdots c}_z$
 $s' = uv^0 xy^0 z \notin C$

② $s = a \cdots ab \cdots b c \cdots c$
 $\underbrace{a \cdots a}_u \underbrace{b \cdots b}_{vx} \underbrace{c \cdots c}_z$
 $s' = uv^0 xy^0 z \notin C$

③ $s = a \cdots ab \cdots b c \cdots c$
 $\underbrace{a \cdots a}_u \underbrace{b \cdots b}_v \underbrace{c \cdots c}_z$

condition 3. $|vxy| \leq p$ 만족 x

1.c : c가 없는 경우

1.a 증명과 동일

Case 2. ret 뒤가 2가지 이상의 alphabet을 가지는 경우

→ abc 순서에 맞지 않게 pumping됨.

⊕ c는 CFL이 아님

Ex 2.38

$$D = \{ww \mid w \in \{0,1\}^*\}$$

① Pumping 가능한 string

$$s = 0^p 1 0^p 1$$

$$= \underbrace{0 \cdots 0}_n \underbrace{1}_{v} \underbrace{0 \cdots 0}_p \underbrace{1}_{z}$$

$$s' = uv^2 xy^2 z \in D \rightarrow \text{증명 실패}$$
$$s'' = uv^0 xy^0 z \in D$$

② Pumping 불가능한 string \rightarrow 존재하기에 잘 선택해서 증명할 것!

$$s = 0^p 1^p 0^p 1^p$$

Case 1. vxy 가 mid point 보다 앞쪽에 존재

$$\begin{aligned} s &= 00 \cdots 011 \cdots 100 \cdots 011 \cdots \\ &\quad \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{\text{mid}} \\ s' &= uv^2xy^2z \\ &= 00 \cdots 011 \cdots 100 \cdots 011 \cdots \\ &\quad \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{\text{mid}} \\ &\quad \hookrightarrow 더 이상 중간이 x \\ &\notin D \end{aligned}$$

Case 2. vxy 가 mid point 보다 뒤쪽에 존재

$$\begin{aligned} s &= 00 \cdots 011 \cdots 100 \cdots 011 \cdots \\ &\quad \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{\text{mid}} \quad \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{vxy} \\ s' &= uv^2xy^2z \\ &= 00 \cdots 011 \cdots 100 \cdots 011 \cdots \\ &\quad \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{\text{mid}} \\ &\notin D \end{aligned}$$

Case 3. vxy 가 mid point 사이에 존재

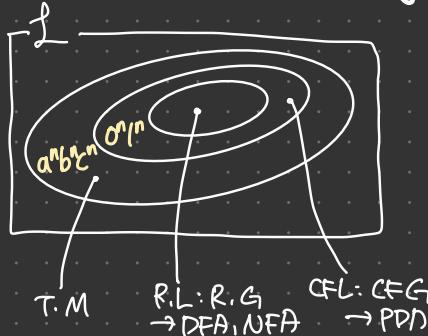
\Rightarrow 앞쪽 w가 뒤쪽 w와 달라짐

$$s' \notin D$$

$\Rightarrow C=lo1 X$

Part Two : Computability Theory

3. The Church - Turing Machine



T.M \Rightarrow PDAes read/write 제약
PDA, NFAes small amount 해결

3.1 Turing Machines

① introduce

→ powerful한 model \rightarrow Alan Turing이 고안한 Turing machine

unlimited and unrestricted한 finite automaton

\rightarrow general purpose computer의 시초

Put. 풀지 못하는 문제 (theoretical problem) 있음 (ex. halting)

① Turing Machine Model

: an infinite tape as its unlimited memory.



\Rightarrow tape : read, write and move 가능 \rightarrow head

i) the tape : input string을 담고 있으되 blank도 가지고 있음.

ii) tape에 information을 write하여 저장

iii) head를 움직여서 쓰레진 information을 읽을 수 있음.

iv) accept, reject \Rightarrow state를 둘 다 생성

\Rightarrow loop forever 가능!!

\rightarrow finite automata와 다른점

- tape에서 read, write 모두 가능
- tape에서 head를 left, right 둘 다 move 가능
- tape가 infinite
- ~~- state가 accept, reject 모두 존재~~

[Ex] $B = \{w\#w \mid w \in \{0,1\}^*\} \rightarrow M_1$

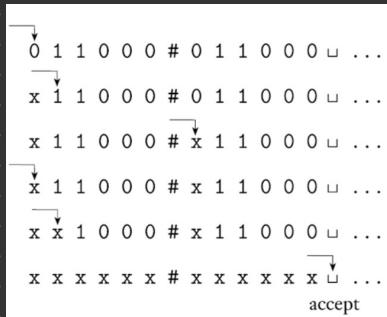
$s = \textcircled{x} \textcircled{1} \textcircled{1} \textcircled{\#} \textcircled{x} \textcircled{1} \textcircled{1}$

\Rightarrow 하위의 head를 움직이면서 #을 기준으로 match 표시

\Rightarrow input을 다 일치하는지 mismatch 되거나 끝나면 reject

"~~없는~~ accept"

$$s' = 011000 \# 011000$$



② Formal Definition of a Turing Machine

f : transition function

$$\Rightarrow Q \times T \rightarrow Q \times T \times \{L, R\}$$

$$f(g, a) = (t, b, L)$$

$\rightarrow a$ 를 b로 치고 state는 원쪽으로

In nondeterministic

$$f \rightarrow Q \times T \rightarrow P(Q \times T \times \{L, R\})$$

Definition 3.3: Turing machine

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

- ① Q is the set of states.
- ② Σ is the input alphabet ~~not~~ containing the blank symbol \square .
- ③ Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$. \leftrightarrow finite: Σ
- ④ $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function. \leftrightarrow P.D.A: stack \rightarrow stack ~~stack~~!
- ⑤ $q_0 \in Q$ is the start state,
- ⑥ $q_{\text{accept}} \in Q$ is the accept state, \checkmark
- ⑦ $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$. $\boxed{\begin{array}{l} \text{매우한 } \checkmark \\ \text{여러개 있어도 Power가 같아서} \\ \text{정상화} \end{array}}$

③ Computation of T.M

i) $M = (Q, \Sigma, T, f, z_0, z_{\text{accept}}, z_{\text{reject}})$

ii) $w = w_1 w_2 \dots w_n \in \Sigma^*$ \rightarrow input tape $\underline{w_1 w_2 \dots w_n \square \square \dots}$

iii) head : 가장 원쪽에서 시작 \rightarrow head $\boxed{\quad}$

iv) \square (blank symbol) $\notin \Sigma$, input의 끝에서 가장 먼저 나타남

v) M이 시작되면 f에 의한 rule대로 proceed.

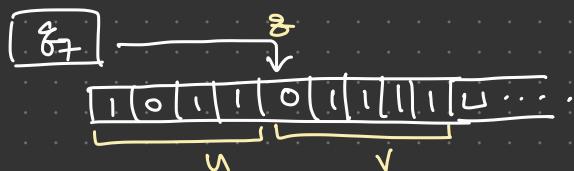
- vi) left의 끝에 >면 (가장 시작으로 가면) 어딨어 할지 하니로 정해지기.
- vii) accept or reject 가능지 정할 때까지 계속 함.
 ↳ 아무것도 결정되지 않으면 forever loop.

④ Configurations of a T.M

↳ Configurations [① current state
 ② current tape contents
 ③ current head location] } 현재 설정의 정확한 TM 모음을 설명

i) configuration : $\boxed{\gamma_0} \rightarrow \text{input alphabet}$
 ↳ state

ex) (0|1|8|0|1|1|)



ii) yield

C_1 에서 C_2 끌어갈 때, C_1 yields C_2 라고 표기

leftward : $a\alpha b\beta c\nu$ yields $a\beta\alpha c\nu \Rightarrow f(\gamma_i, b) = (\gamma_j, c, L)$

$$(a, b, c \in \Gamma, u, v \in \Gamma^*)$$

rightward :

$a\alpha b\beta c\nu$ yields $a\alpha c\beta b\nu \Rightarrow f(\gamma_i, b) = (\gamma_j, c, R)$

left-hand end :

$\gamma_i b \nu$ yields $\gamma_j c \nu$ (leftward)

yields $c\gamma_j \nu$ (rightward)

right-hand end :

$$w\#z = w\#z\Gamma$$

start configuration : $\$ \# w$

accepting configuration : state $\Rightarrow \text{accept}$

rejecting configuration : state $\Rightarrow \text{reject}$

halting configuration $\rightarrow \text{yield}$ 블가

$$\hookrightarrow f: Q^* \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

\hookrightarrow Q에서 accept, reject 처리

⑤ The Language of TM

: M accepts input w 일 때

w의 configuration은 C_1, C_2, \dots, C_k 라고 하자.

i) C_i : start configuration

ii) C_i yields C_{i+1}

iii) C_k : accepting configuration

\Rightarrow 모두 만족하는 string들의 집합 = $L(M)$

(PDA, NFA²랑 다르게 accept, reject, forever loop)

\hookrightarrow string은 3가지로 분류

⑥ Turing recognizable

: T.M이 recognize하는 Language \rightarrow Turing - recognizable

i) T.M은 OUTPUT이 3가지 \rightarrow accept, reject, loop

But, halt 불가 \rightarrow loop인지 찾아내는게 꽤 어렵음.

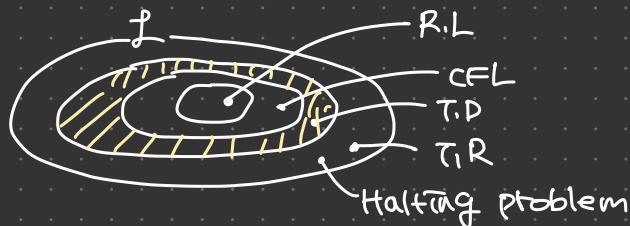
ii) 주어진 T.M이 loop를 하는 경우를 하나라도 가질는지 판별하고 싶음.

\Rightarrow 되면 Turing decidable

accept, reject \leftarrow
들 속에 허락 \Rightarrow decide

⑦ Turing - decidable

: accept, reject 둘 중에 무조건 결정하는 string만 recognize



[Ex 3.7] M₂ decide $A = \{0^n \mid n \geq 0\}$

① 2개의 전처리에서 0을 check하는 것을 반복

② 0이 하나 \rightarrow accept

③ 0이 두 개 이상인경우 \rightarrow reject

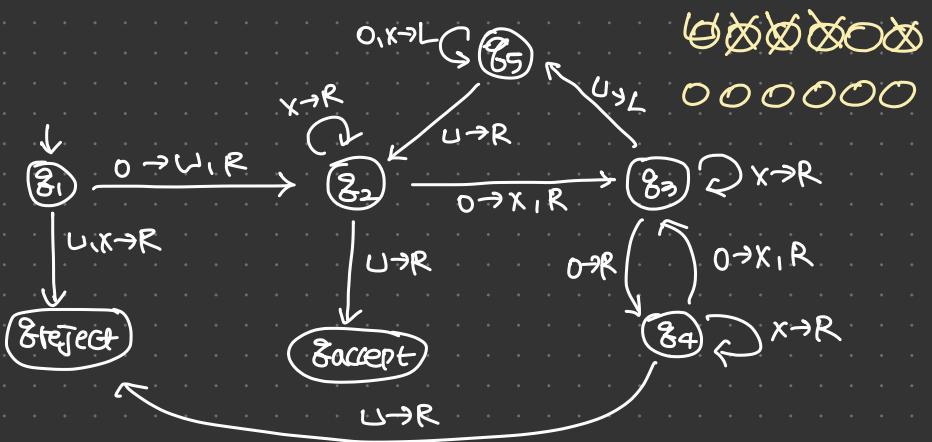
④ head가 원쪽 끝에 도달하면 return

⑤ ①② 돌아감

$$M_2 = (Q, \Sigma, T, f, g_1, g_{accept}, g_{reject})$$

$$Q = \{q_1, q_2, \dots, q_5, q_{accept}, q_{reject}\}$$

$$\Sigma = \{0^3, T = \{0, X, \sqcup\}$$



8.2 : 첫번재 0을 U로 만들고 시작

8.3 : 짝수개만 남도록

⇒ 한 번 들면서 짝수이지만 그의 제곱이 아님 걸 찾음

$q_1 0000$	$U q_5 x 0 x u$	$U x q_5 x x u$
$U q_2 000$	$q_5 u x 0 x u$	$U q_5 x x x u$
$U x q_3 00$	$U q_2 x 0 x u$	$q_5 U x x x u$
$U x 0 q_4 0$	$U x q_2 0 x u$	$U q_2 x x x u$
$U x 0 x q_3 u$	$U x x q_3 x u$	$U x q_2 x x u$
$U x 0 q_5 x u$	$U x x x q_3 x u$	$U x x q_2 x u$
$U x q_5 0 x u$	$U x x q_5 x u$	$U x x x q_2 u$
		$U x x x u q_{\text{accept}}$

[Ex 9.9] M_1 decide $R = \{ w\#w \mid w \in \{0,1\}^*\}$

$$Q = \{q_1, \dots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$$

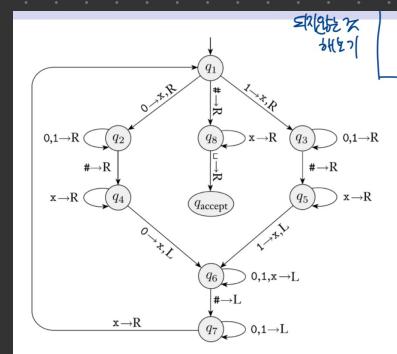
$$\Sigma = \{0, 1, \#\}$$

$$\Gamma = \{0, 1, \#, X, U\}$$

i) reject state는 생략

c) 8단에는 #를 INPUT으로 받는

arrow 짼 X \Rightarrow reject



Ex 3.11 M₃ decide $C = \{a^i b^j c^k \mid i \neq j \neq k \text{ and } i, j, k \geq 1\}$

- i) input을 처음부터 끝까지 scan하면서 $a^i b^j c^k$ 순서인지 check
⇒ 아니라면 reject
- ii) 가장 왼쪽까지 온 다음에 안 쓰는 value로 표시
- iii) a를 하나 치우고 b는 다 치운 뒤 b를 치운만큼 c를 치운
⇒ C를 치우다가 b가 없으면 reject
- iv) 치운 b를 다시 살린 다음에 ③ 반복
a를 다 치웠는데 C도 다 치웠다면 accept

⇒ i)은 finite automata처럼 작동

∴ 서서 저작할 필요 없이 알기만 하면 됨

⇒ ii)는 간단하지만 미묘하게 어려움

↳ leftmost 어려워 표시?

: **Ex 3.7**처럼 machine!

Ex 3.12 M₄ → element

$E = \{ \#x_1 \#x_2 \cdots \#x_l \mid \text{each } x_i \in \{0, 1\}^* \text{ and } x_i \neq x_j, i \neq j \}$

ex. #011#100#10

⇒ x_1 을 $x_2, x_3, x_4, \dots, x_l$ 까지 빼고

x_2 을 x_1, x_3, \dots, x_l 까지 빼고

⋮

x_l 을 x_1, x_2, \dots, x_{l-1} 까지 빼고

ii) 지금 빼고하고 있는 string 표시  #011#100#1

만약 blank → accept

만약 # → 다음 #로 넣어갈, #가 아니면 reject.

v) # 다음에 있는 #에도 mark

만약 #이 있고 아직 첫번째 string만 있다면 accept.

vi) 표시한 # 모른쪽에 있는 string들을 각각 하나씩 비우

만약 같으면 reject

vii) 왼쪽의 mark를 다음 왼쪽으로 킥김.

모른쪽 marks 다음 모른쪽으로 move.

만약 # 이전의 blank 있다면 한 칸 뒤로 킥김.

이 비교할 때는 string이 없다면 비우는 → accept.

viii) 3~5 번복

⇒ marking of tape symbols 사용. (# → #)

↳ 모든 symbol에 적용 가능

정을 추가한 버전을 각 alphabet에 추가할 것

↳ loop forever 블록은 사용 X

3.2 Variants of Turing Machines

↳ multiple tapes / nondeterminism

⇒ The original model과 같은 power가 같음.

⇒ 다양한 variants And 어떤지 증명!

① Preview

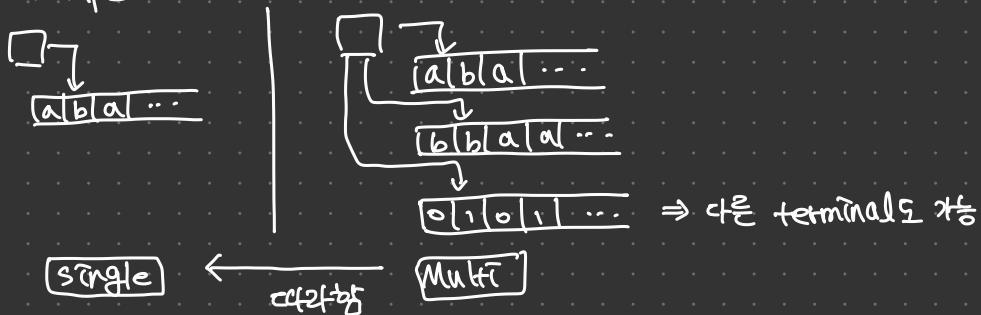
i) TM의 대량 definition → left, right, stay put

ii) 이 특징이 TM이 인식하는 language를 recognize 할지도?

↳ 아니! left, right transition만 존재

ii) 가능이 많은 TM' : 가능이 적은 TM을 따라남. \Rightarrow power가 같음.

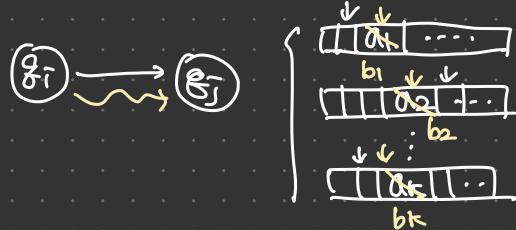
① Multitape T.M



i) Transition function $\Rightarrow f : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

$k = \text{the number of tapes}$

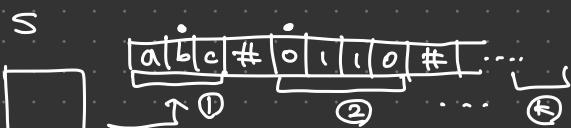
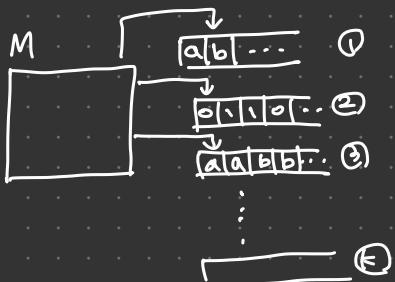
$$\Rightarrow f(q_i, a_1 \dots a_k) = (q_j, b_1 \dots b_k, L, R, \dots, L)$$



ii) [Th 7.13] Every multitape T.M has an equivalent single-tape T.M

M : k 개의 multitape T.M how to simulate?

S : single-tape T.M



tape \Rightarrow 하나의 tape에 k 개의 tape를 저장
 $\Rightarrow \#$ 으로 구분자 지정

head \Rightarrow multitape에서 head가 가지고 있는 매를 mark.

$$a \rightarrow i$$

\rightarrow 가장자리 tape의 head를 구현

pf) $S = "On input w = w_1 w_2 \dots w_n"$

i) S 는 M 의 모든 헤더 tape을 S 의 tape에 넣음

$$\rightarrow \# w_1 \dots w_n \# \text{L} \# \text{U} \# \dots \#$$

ii) S 가 첫 번째 $\#$ 을 읽으면 가장 왼쪽과 가장 오른쪽에 mark.
(\leftarrow (번갈아))

$\rightarrow M$ 의 transition function이 선택되는 방식에 따라

S 의 tapes update해야 함.

iii) S 의 head가 오른쪽 $\#$ 위에 가면 M 은 하나의 tape의 끝

\rightarrow 오른쪽 $\#$ 전에 blank를 넣어 맞아야 함.

$$\#0010\#abc\dots \rightarrow \#0010\text{L}\#\text{abc}\dots$$

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

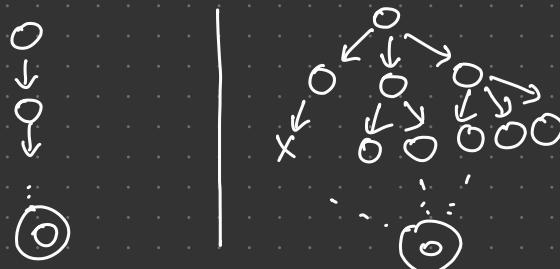
\Rightarrow 반복!

iii) Corollary 3.15 T.R \longleftrightarrow multi tape TM

T.R \rightarrow single tape \rightarrow multi tape

② Nondeterministic TM

: the machine \rightarrow 여러 가지 가능성으로 proceed



i) transition function $\Rightarrow f: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$

\hookrightarrow branch 중 하나로 accept될 accept

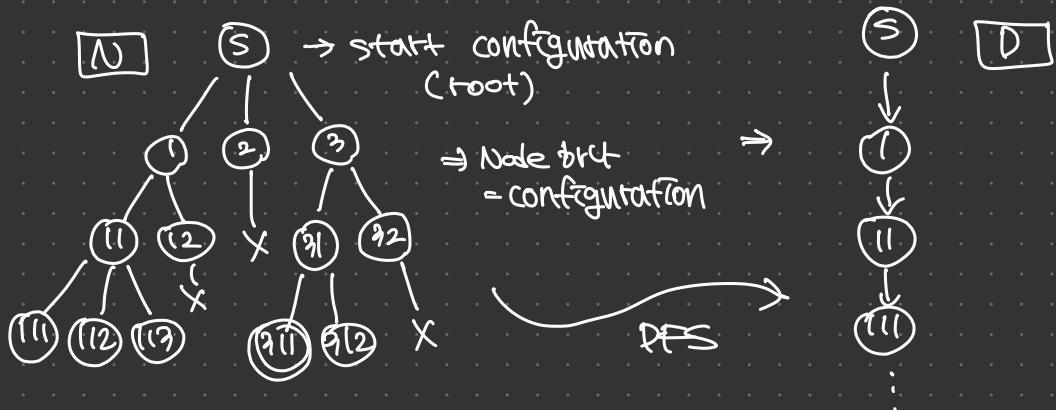
\hookrightarrow 가능한 각각의 가능성들

ii) Th 3.16 Every NTM has an equivalent deterministic TM

N : nondeterministic TM \hookrightarrow how to simulate?

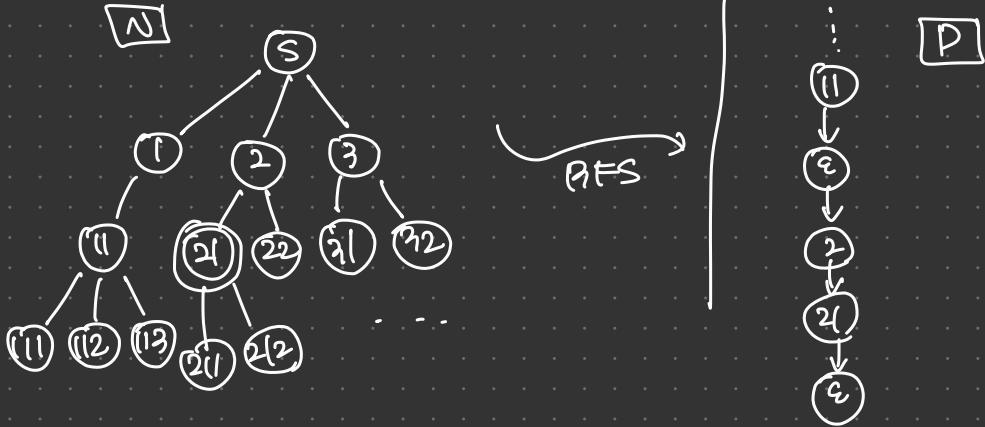
D : deterministic TM

\hookrightarrow D는 하나의 branch이며 accept될 도달하면 accept



D : DFS 2 tree 탐색 \rightarrow bad Idea.

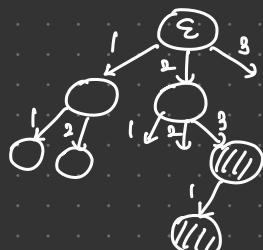
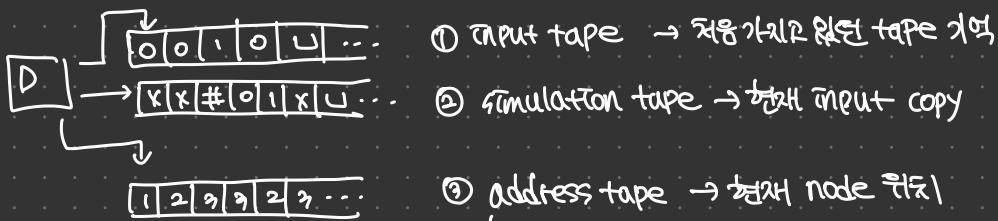
\hookrightarrow 이렇게 생성하면 무한 생성될 것이고 accepting을 놓칠 것임.



D: BFS로 탐색 \rightarrow 같은 깊이에 있는 모든 branch 탐색 후
다음 깊이로 이동
 \rightarrow accepting configuration 찾을 수 있음!

Pf) D : three tape

\hookrightarrow Th 3.17에 의해 single-tape와 equivalent.



$\hookrightarrow 23|2 \text{ 표시}$

*root는 1로 표시

가능한 선택의 차이점 $\Rightarrow b$
+tree의 모든 node $\Rightarrow b$

$$\text{ 모든 node는 } T_b = \{1, 2, \dots, b\}$$

\downarrow

address set

but, T_b 에 속하지만 tree에
들어가지 않을 수 있음.
 $\Rightarrow \text{de} / \text{skiped}$ 간의

i) tape 1 \rightarrow input w
tape 2, tape 3 \rightarrow empty) 초기화

ii) tape (을 tape 2에) 복사

iii) tape 2 : input w에 대한 initial branch를 사용

↳ 연습장처럼 사용

↳ 각 NEL branch를 찾기 이전에 NEL transition function이 elab

만들어지는 choice를 tape 2에 다음 symbol consult.

↳ 만약 accepting configuration이면, accept
(start가 accept)

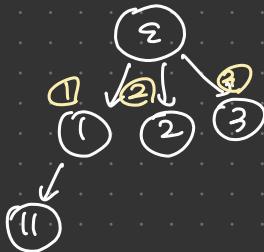
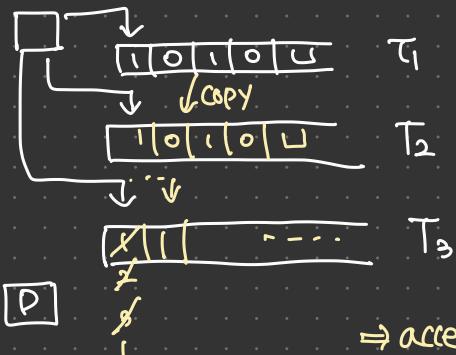
↳ 만약 tape 2에 같은 symbol이 없거나 NEL choice가 없다는
4번으로 넘어감.

↳ rejecting configuration이면 4번으로 넘어감.

(v) tape 2를 BST처럼 다음 step으로 바꿈.

\Rightarrow stage 2를 가서 한복!

w = 1010



\Rightarrow accept or BST 끝날 때까지 검사

iii) Corollary 3.18 T.R \leftrightarrow NTM

Pf) Any DTM \rightarrow NTM

\Rightarrow Th 3.16

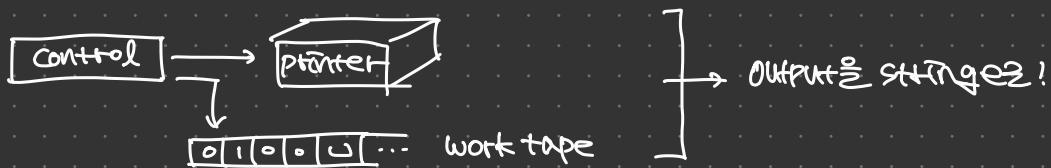
\Rightarrow T.R \leftrightarrow NTM \Rightarrow N.E. 정상 halt \Rightarrow decider

iv) Corollary 3.19 T.D \leftrightarrow NTM

③ Enumerators

T.R \rightarrow recursively enumerable language 라고도 부름

\hookrightarrow 이 language에 대한 T.M variant



만약 enumerator가 halt 되지 않으면 무한한 길이의 string 출력

Th 3.21 language E : print out하는 모든 string들의 집합

TR iff E enumerate.

\Rightarrow step the proof.

\therefore 모든 T.M의 variant는 같은 필수적인 특성을 가지고 있음.

\Rightarrow equivalent in power

3.3 the definition of Algorithm

: a collection of simple instructions for carrying out some task.
an important role in mathematics.

① Hilbert's Problem

i) polynomial : sum of terms
 $\left(\begin{array}{c} \text{항} \\ \text{항수} \end{array} \right)$ $\xrightarrow{\quad}$ variable $\left(\begin{array}{c} \text{변수} \\ \text{변수} \end{array} \right)$ \times product + coefficient
 $\left(\begin{array}{c} \text{제수} \\ \text{계수} \end{array} \right)$ $\left(\begin{array}{c} \text{계수} \\ \text{계수} \end{array} \right)$
 \Rightarrow 모든 coefficient : integer

\Rightarrow root of polynomial : polynomial이 갖는 variable 값.

\Rightarrow 어떤 polynomial은 정수근을 가지고 있음을 알 수 있음.

ii) Hilbert의 10번째 문제는 정수근을 가지는지, 아닌지를 decide 하는 test하는 algorithm.

\Rightarrow 만약 그 어떤 알고리즘도 없다면 unsolvable

\Rightarrow unsolvable한 알고리즘도 있다고 증명!

iii) Church-Turing thesis

\downarrow \downarrow
 λ -calculus \rightarrow T.M \Rightarrow intuitive notion
of algorithms equal T.M

iv) Kurt Gödel, Martin...

: polynomial이 정수근을 가지는지 아닌지를 testing하는 algorithm은 결재하지 않는다라고 증명

PF) $D = \{P \mid P : \text{정수근을 갖는 polynomial 집합}\}$

a. D 가 decidable?

$\leftrightarrow D$ 가 T.R인지 증명

i) $D_1 = \{P = 4x^3 - 2x^2 + x - 7 \text{ 이라고 가정}\}$

M_1 recognizes D_1

$M_1 = \text{"The input is a } p \text{ over the variable } x\}$

$\left[\begin{array}{l} p \text{가 } x = 0, 1, -1, -2, 2, \dots \text{ 중 어느거나} \\ \text{0이 되면 accept} \end{array} \right]$

만약 정수근을 가지지 않는다면 forever loop.

ii) M recognizes D

M 은 모든 정수값에 대해서 가능한 getting.

M_1 을 D_1 의 decider로 변환 \rightarrow bound 존재함!

$\left[\begin{array}{l} \text{toot가} \\ \pm c_{\max} \end{array} \right] \rightarrow \text{계수 중 최댓값.}$

c_i 사이에 있음!

방의 개수 \rightarrow 최고차항 계수

\rightarrow 만약 toot가 이 범위 내에 존재하지 않으면 reject

iii) But. 계수 안되는거이잖아

(multivariable polynomial은 끝나지.)

3.4 Terminology for Describing T.M

T.M \rightarrow Algorithm

↳ 너무 low level Algorithm 말고 TM이 풀만한 Algorithm.

\Rightarrow 어떤 level? TM이 describable 하는 Algorithm?

\Rightarrow 3가지 조건

- ① formal description \rightarrow state, transition, ... (detail)
- ② implementation description \rightarrow head, tape ...
- ③ high-level description \rightarrow English prose
(tape, head 연습 X)

\Rightarrow 지금까지 수준 예제만 해왔으나 high level도 해보자.

하기에 먼저,

* notation

input \rightarrow 정상 string \rightarrow object로 표현

ex. $O_1, O_2, O_3 \dots, O_k \Rightarrow \langle O_1, O_2 \dots O_k \rangle$ 표현

object 내용은 accept, reject 결정

Ex 3.23

A : connected vertex 없는 undirected graph를 나타내는
B의 string을 포함하는 language

\rightarrow B의 nodes를 traveling하면서 모든 다른 nodes에 접근 가능.

$A = \{ \langle G \rangle \mid G \text{ is a connected undirected } \}$

i) High-level description : TM M decides A

M = "On input $\langle G \rangle$, the encoding of a Graph G :

- ① G 의 첫 번째 node 선택 후 mark
- ② 다음 node가 mark된지 않은지 대기
- ③, ④ 반복
- ⑤ 모든 edge \rightarrow 0이 mark된 node에 연결되어 있는 모든 edge mark.
- ⑥ 모든 node \rightarrow mark가 되었는지 scan 만약 다 되면 accept 아니면 reject

ii) Implementation level

① string G 를 어떻게 $\langle G \rangle$? Encoding?



$$\langle G \rangle = \text{node } 1, 2, 3, 4 | \text{edge } ((1,2), (2,3), (3,1), (1,4))$$

② M이 Input $\langle G \rangle$ 을 받았을 때
Graph와 함께 Encoding 유효하는지 check
 \Rightarrow tape를 scan하면서 (1st 2nd 3rd 4th)



10진수
node



pair of 10진수
edge

a. node list : 중복 있는지 check
모든 node가 edge (list)에 있는지
 \Rightarrow [Ex 3.12] 치즈 check

b. edge list 도 동일

③ input이 이 과정 다 통과하면 Encoding됨.

(ii) stage 1 : Mol 첫번째 node (leftmost) 표시 *

Stage 2 : Mol undotted node n_1 에 표시 (-)

Mol dotted node n_2 에 표시 (-)

Stage 3 : edge scan

n_1 과 n_2 edge 조사(하나)
 n_1 의 (-) mark 지우고 Stage 2

없으면 다음 edge check

check 하면 n_2 를 다음 dot node? more

\Rightarrow dotted node인 경우 Stage 4

Stage 4 : 모든 모든 node가 dotted인지 표시

맞다면 accept / 아니면 reject

Part Two. Computability Theory

4. Decidability

: to investigate the power of algorithms to solve problems.

⇒ unsolvability에 대해 공부

① unsolvable를 아는 것은 유용한

② 풀리지 않는 거라면 풀 시도? ..

4.1 Decidable languages

: algorithm이 의해 decidable한 language 예시를 살펴볼 것!

⇒ language가 automata, grammar에 연관이 있는지 focus

ex. 어떤 string이 주어진 R.L의 member인지 test하는 알고

① decidable problems concerning R.L

FAP인 B_1 , $\text{att } w \Rightarrow \text{accept 하는지를 Algorithm/TM에 대입!}$

i) acceptance problem for DFA

주어진 string을 특정한 DFA가 testing하는지 $\rightarrow A_{DFA} \ni \text{표현}$

ii) $A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \}$

\Rightarrow showing A_{DFA} is decidable

= Computational problem is decidable!

[Th 4.1] A DFA is a decidable language.

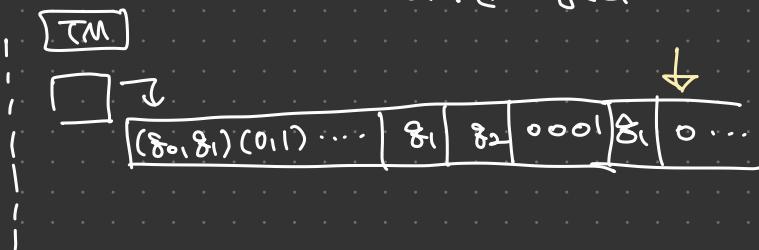
Pf Idea) M decides A DFA

M = "On input $\langle B, w \rangle$, where B is a DFA and w is a string."

- ① Simulate B on input w
- ② If simulation on accept state 끝나면 accept.
아니면 reject.

Pf) ① $\langle B, w \rangle$
Implementation Level $\hookrightarrow Q, \Sigma, \delta, q_0, F$

- ② M이 input을 받으면 가장 먼저 M이 B가 w를 정확하게 represent하는지 여부를 determine
아니면 reject
- ③ M: simulation directly
 \Rightarrow B의 current state와 tape의 마지막에 정보를 writing하면서 input positions도 찾아감.
 \Rightarrow for accept update.
- ④ M이 w의 last symbol을 읽고 processing을 끝날 때,
M은 B가 accepting state에 있으면 accept
아니면 reject



iii) $\text{ANFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \}$

[Th 4.2] ANFA is a decidable language.

Pf) TM N decides ANFA

→ N이 M(A DFA)을 subroutine으로 가진다고 생각

$N = \text{"On input } \langle B, w \rangle \text{ where } B \text{ is an NFA,}$
and w is a string "

① NFA B 를 equivalent한 DFA C 로 바꿈.

② TM M 을 [Th.4.1]의 input $\langle C, w \rangle$ 로 실행

③ M이 accept하면 accept, otherwise reject.

→ subprocedure.

iv) $\text{EDFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \}$

→ TM이 할 수 있는가?

→ empty language

→ 어떤 string
accept state X

[Th 4.4] EDFA is a decidable language

Pf) A DFA accepts some str \leftrightarrow startofA acceptofA 도달하는
DFA의 arrow가 있다면

→ arrow 없는지 확인하는 marking algorithm으로 TM T design.
(B는 state check하면서 accepter 연결된 arrow 찾기)

$T = \text{"On input } \langle A \rangle \text{ where } A \text{ is a DFA}$

① mark the start state of A

② mark되지 않은 새로운 state가 없을 때까지 반복

③ all marked state를遍历하는 transition이 없는 state

⊕ 만약 match 된 accept state가 - 같으면 accept
아니면 reject

v) $\text{EQDFA} = \{ \langle A, B \rangle \mid A, B \Rightarrow \text{DFAs} \text{ and } L(A) = L(B) \}$

[Th.4.5] EQDFA is a decidable language.

Pf) [Th.4.4] pf 이祟

$A \cap B \rightarrow$ 새로운 DFA C 를 construct.

C : either A or B 만 accept



만약 A et B 가 같으면 C 는 아무것도 accept x

$$\Rightarrow L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

$$\Rightarrow L(C) = \emptyset \text{ iff } L(A) = L(B)$$

$\Rightarrow C$ 는 A et B 의 regular operation에 展성 가능

$\Rightarrow C \neq \emptyset$ 인지 [Th.4.4] 를 이용하여 proof

$$(L(A) = L(B))$$

$F =$ "On input $\langle A, B \rangle$, where A and B are DFAs :

① Construct DFA C as described.

② Run TM T from [Th.4.4] \rightarrow input $\langle C \rangle$ 에 대해

③ 만약 $T \rightarrow \text{accept} \rightarrow C \neq \emptyset \rightarrow L(A) = L(B)$
 $\Rightarrow \text{accept}$

" reject \rightarrow reject

② Decidable Problems Concerning CFL

i) $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \}$

Th 4.7 A_{CFG} is a decidable language.

Pf Idea) CFG : $G \quad \rightarrow$ G 가 w 를 generate하는지?
str : w

$\Rightarrow G$ 의 derivation 중 w 의 derivation이 있는지?

\hookrightarrow But, 여러 가지가 나올 수 있어 halt X

\Rightarrow TM이 recognize but decider X

\Rightarrow bound가 정해지면 decider가 될 수 있음.

(finite한 derivation)

\Rightarrow Chomsky Normal form : w 은 $\frac{(2n-1) \text{ derivation}}{n: w \text{의 길이}}$

$\therefore G$ 가 $(2n-1)$ 개의 derivation으로 w generate 가능하지
check!

Pf) TM S = "On input $\langle G, w \rangle$, where G is a CFG
and w is a string."

① G 를 chomsky Normal form

② 모든 $2n-1$ step의 derivation을 나열

만약 $n=0 \rightarrow$ step이 하나인 derivation 나열

③ 이 중 하나라도 w 를 generate하는 accept
하지 않으면 reject.

Th 4.9 Every CFL is decidable

Pf Idea) A가 CFL이면 증명하자.

Bad : A를 PDA로 convert하고 TM와 연결하는 건 bad

(\because stack, tape convert하기 어려움)

NPDAs도 마찬가지, loop forever \Rightarrow halting X

\Rightarrow TM Set Th 4.7의 Acfg로 증명

Pf) CFG : G for A . \hookrightarrow copy!
TM M_G decides A

$M_G =$ "On input w :

① TM S $\langle G, w \rangle$ inputs Run

② If accept \Rightarrow accept / reject \Rightarrow reject.

4.2 Undecidability

: computer도 모든 문제를 다 풀 수 있는 건 X

\Rightarrow unsolvable한 문제에 대해 알아보자.

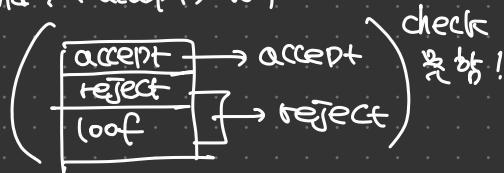
① first theorem : TM이 주어진 input set을 accept하는지 determining하는 문제. \Rightarrow ATM.

\Rightarrow AFA, Acfg는 decidable

But, ATM은 undecidable

i) $A_{\text{ATM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$

Th 4.11 ATM is undecidable.



① ATM은 T.R. ?

T.M U recognizes ATM

$U = \text{"On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ is a str."}$

① simulate M on Input w.

② M이 accept state에 들어가면 accept.

"reject" "reject"

$\Rightarrow U$ 가 $\langle M, w \rangle$ input을 때永远 loop면 U 가 loop.

$\Rightarrow U$ 가 ATM decide ∞ X

② 만약 이 Algorithm이 M이永远 halt ∞ 인지 아닌지 determine 해주는 방법을 가지고 있다면 loop일 때 reject.

\therefore Put, 이런 Algorithm X

② Universal T.M

: stored-program computer의 이전 version.

program M들을 받아서 그것을 수행

i) the diagonalization method.

: ATM의 undecidability를 사용하는 기술

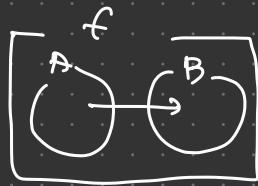
\Rightarrow not-decidable (But) recognizable

\Rightarrow 이러한 문제의 infinite set의 크기를 측정하는 문제.

\Rightarrow 둘의 크기는 같은가? 아니면 하나가 더 크다?

\Rightarrow infinite set에 가수를 세지 않고 하나씩 대응하여 크기 \Rightarrow check.

Def 4.12



f : function $A \rightarrow B$

A, B : sets

① f : one-to-one (일대일)

② f : onto $\rightarrow f(a) = b$

$\Rightarrow f: A \rightarrow B$

정의에 의한 두 가지 치역의 한 원소와 대응

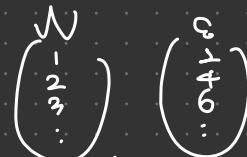
Ex 4.13

N : 자연수 집합 / E : 짝수 집합

$f: N \rightarrow E$

$$f(n) = 2n$$

\Rightarrow 직관적으로 E 가 적아보임.



Put \forall 모든 member를 E 에 대응할 수 없기 때문에
두 set이 same size.

Def 4.14

\exists countable \rightarrow 유한하거나 N 와 같은 것 같거나

Ex 4.15

$Q = \{ \frac{m}{n} \mid m, n \in N \} \Rightarrow$ 양의 유리수 집합

\hookrightarrow N 보다 size가 클 것 같음 \Rightarrow but same!

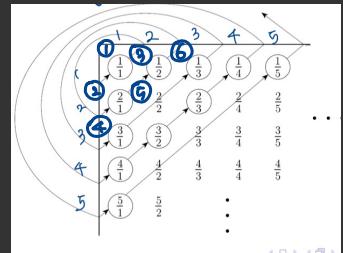
① Q is countable

\Rightarrow \exists correspondence 가능가?

\Rightarrow infinite matrix로 표시

row, col이 끊어지지

안도록 diagonal!



$$\Rightarrow \frac{1}{1}, \frac{2}{2}, \frac{3}{3} = 1 \text{인 } N \text{에 정의역에서 제외}$$

② Ω 가 모든 N과 correspondence.

$\therefore \Omega$ 은 same size

[Th4.17] \mathbb{R} is uncountable

0) uncountable set $\rightarrow \mathcal{U}, \Omega, \mathbb{R} \rightarrow$ ~~size~~

1) \mathbb{R} is uncountable \rightarrow N과 R의 correspondence X

By contradiction, correspondence f 가 존재한다고 가정

\hookrightarrow R의 모든 member와 N의 모든 member가 짝이 있어야 함.

\Rightarrow R에 있는 x가 N에 없다는 걸 증명

2) $x \rightarrow N$ 의 각자리 숫자와 다르게 치적

ex)	n	$f(n)$	$(0 < x < 1)$
	1	3. 1 4199...	$x = 0.23\overline{419}$...
	2	55. 5 555...	
	3	0. 12 3 55...	\downarrow
	4	0. 500 0	다른게 선택하다 보면 같은 숫자가 나을 수가 X
	:	:	

$\therefore \mathbb{R}$ 은 uncountable

(N과 모두 대응 X)

· diagonalization method :

: T.R의 개수 $>$ T.M의 개수

(uncountable) (countable)

\Rightarrow T.M이 ~~able~~ recognize 되지 않는 (language) \Rightarrow A



이런 problem
존재 증명



Corollary 4.18 some languages are not T.R.

① 모든 TM이 countable

→ 모든 Σ에 대한 $\text{G}(T)$ Σ^* 의 집합은 countable

→ length에 따라 가능한 $\text{G}(T)$ list 가능

(→ TM은 하나의 Encoding string을 가짐)

⇒ 모든 TM 집합은 $\langle M \rangle$ 으로 Encoding한 $\text{G}(T)$ 가 countable!

$$\langle M \rangle = "Q, \Sigma, T \dots"$$

(TM 형식이 아닌 것도 있지만 일단 대응 가능)

② 모든 language 집합이 uncountable

→ 모든 무한한 binary sequence가 uncountable
하다는 것을 증명 ↳ 0101...

① B : the set of all infinite binary sequences.

② B is uncountable

i) B 가 countable이라고 가정

ii) $N \rightarrow B$ correspondence f 가 존재

N	f	$n=100\dots$
1	0010...	0000...
2	0100...	0000...
3	0011...	0000...
⋮	⋮	⋮

$f(n)$ 과 대응하지 않는
 $x \in B$ 존재

③ L : the set of all languages over Σ .

iii) $B - L$: correspondence 존재 가정

ex) $\Sigma^* = \{ a_1, a_2, a_3, \dots \}$

각 language $A \in \mathcal{L}$

↳ B의 sequence 중 하나

$x_A \Rightarrow$ i 번째가 A에 속하면 1, 아니면 0

ex) A : 0으로 시작하는 set

$\Sigma^* = \{ \epsilon, 0, 1, 00, 01, 10, 11, \dots \}$

language $\rightarrow A = \{ \underset{\downarrow}{1}, \underset{\downarrow}{0}, \underset{\downarrow}{1}, \underset{\downarrow}{00}, \underset{\downarrow}{01}, \underset{\downarrow}{10}, \underset{\downarrow}{11}, \dots \}$

binary sequence $\rightarrow x_A = \{ 0, 1, 0, 1, 1, 0, 0, \dots \}$

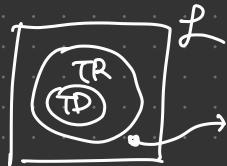
$$= \overbrace{0101100 \dots}^{\text{set } A \text{에 대응}}$$

↳ set A에 대응

$\therefore f: \mathcal{L} \rightarrow \mathcal{B} \Rightarrow$ correspondence.

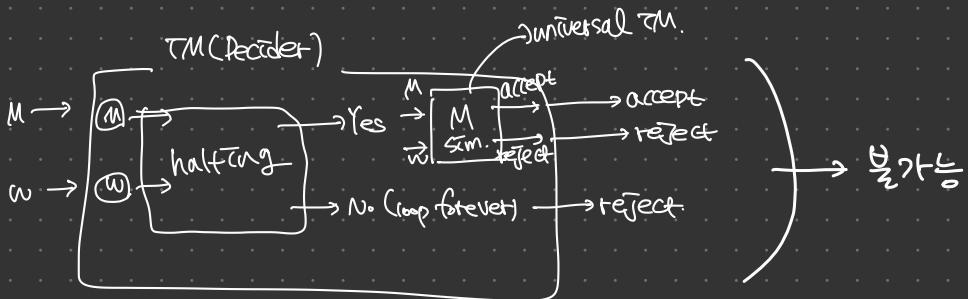
$\Rightarrow \mathcal{B}$ 가 uncountable $\rightarrow \mathcal{L}$ 도 uncountable

$\Rightarrow \mathcal{B} \subseteq$ language를 모든 T.M에 대응 X
(T.M은 countable)



Halting problem

2차 학습을 끝에 없음.



③ Halting Problem

Th 4.11) → ATMOL undecidable \Rightarrow 증명

i) Halting Problem

- : loop forever 여부를 finite time에 증명할 수 있는 decidable한 algorithm은 존재하지 X

pf by contradiction)

Input: <program p>, <input t>

Output
[true : p가 t를 입력 받아서 실행이 종료하는 경우
[false : p가 t를 입력 받아서 종료 X, loop

The halting problem is undecidable.

- 아래와 같이, 그 내부에서 halt를 호출하는 프로그램 trouble를 생각해보자.

```
1: function trouble(string s){  
2:   if (halt(s, s) == false)  
3:     return true;  
4:   else  
5:     loop forever; // while(1){}; b/c.  
6: }
```

- 모든 프로그램을 string으로 표현할 수 있으므로, 위의 프로그램 trouble을 string t로 변환했다고 하자.

- 이 string t를 프로그램 trouble의 입력으로 하는, trouble(t)의 실행 결과에 대해서 생각해보자.

1) trouble(t)가 halt하려면, line 2에서 halt(t, t)가 false를 return해야 한다. 그러나, halt(t, t)가 false를 return하려면, halt 프로그램의 설명 (2)에서처럼 프로그램 t가 t를 입력으로 받아 그 실행이 loop forever해야 한다. 즉, trouble(t)가 loop forever해야 한다. 모순.

2) trouble(t)가 loop forever하려면, line 2에서 halt(t, t)가 true를 return해야 한다. 그러나, halt(t, t)가 true를 return하려면, halt 프로그램의 설명 (1)에서처럼 프로그램 t가 t를 입력으로 받아 그 실행이 종료해야 한다. 즉, trouble(t)가 종료해야 한다. 모순.

\rightarrow program <p>, <t>는 종료 X \Leftrightarrow 결정이 불가능임

\Rightarrow halt는 존재할 수 X

ii) [Th 4.11] ATM is undecidable

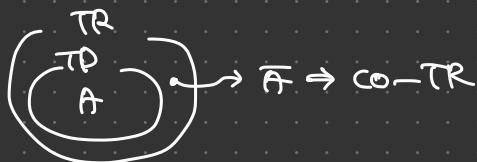
① A ∉ T.R 도 있음

↳ $A, \overline{A} \notin T.R \Rightarrow$ decidable

② undecidable language

↳ $A : TR \& \overline{A} : TR \rightarrow A : TD$

$\Rightarrow \cancel{A : TD} \rightarrow \cancel{A : TR}$ or $\cancel{A : TR}$



iii) [Th 4.22] A Language is decidable \leftrightarrow TR, co-TR

pf \rightarrow) 위에서 증명

pf \leftarrow) $A \rightarrow$ recognize M_1 ,
 $\overline{A} \rightarrow$ " M_2

$\Rightarrow A$ 에 대한 decider M

$M =$ "On Input w : (two tape)"

① M_1 & M_2 \models w 에 대한 동일한 수행

② M_1 이 accept \rightarrow accept

M_2 가 accept \rightarrow reject

$\Rightarrow M_1$ 이나 M_2 가 연습가 한 번이 accept

\Rightarrow 항상 M 은 halt \Rightarrow decider!

$\therefore M$ 은 A 에 대한 decider $\rightarrow A$ is decidable

[Corollary 4.23] $\overline{\text{ATM}}$ is not T.R

$\overline{\text{ATM}} \in \text{T.R}$ 이므로 ATM 은 decidable 이다.

But. [Th 4.11]により ATM 은 decidable X

Theory of Computation ← Gödel!

- Automata theory
- Languages (problem) class

$\text{L}(A) > \text{TR}$

FIGURE 4.10
The relationship among classes of languages

