Chap. 5
Context-free Languages

Agenda of Chapter 5

Compiler. parsing [illegible]

- Context-free Grammars
  - Examples of context-free languages
  - Derivation trees
- Parsing and Ambiguity
- Context-free Grammars and Programming Languages

production의 형태에 대해

Examples of context-free languages(1/3)

[Definition] Context-free grammars G=(V, T, S, P)

A grammar with all productions of the form,

$A \rightarrow x$

where $A \in V$ and $x \in (V \cup T)^*$

Variable 1개.

Variable, Terminal 섞어서 x도 됨.

aA→axy

무조건 A 뒤에 a가 있어야 바뀔 수 있기 때문에 Context free가 아니다.

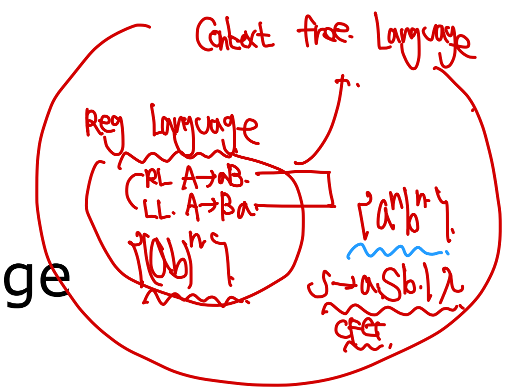
[Definition] Context-free languages

L is context-free ⇔ there is a CFG G such that L=L(G)

linear-grammar 랑 일때. (variable 1개일때)

Terminal symbol 이 양쪽으로 가능

- $F_{RL} \subset F_{CFL}$
  - F_{RL}: A family of regular language
  - F_{CFL}: A family of context free language



- Meaning of "Context-free" →문맥에 상관 없다
  - A→x.
  - production의 왼쪽에 있는 variable은 sentential form에 나타날 때마다, 나머지 부분에 상관없이 대체 가능.

x도. (그 양쪽에 뭐가 있든지)

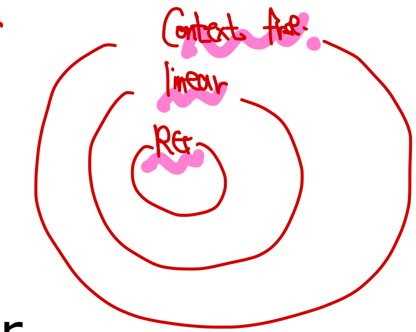
Examples of context-free languages(2/3)

Ex5.1] G=({S},{a,b},S,P) with

- S → aSa, S → bSb, S → λ
- G is context-free and linear, but ~~not regular~~.
- L(G) = $\{ww^R \mid w \in \{a,b\}^*\}$

→ Context free Grammar but not Regular Grammar.

$S \to aSa \Rightarrow abSba$



Ex5.2] G with productions

- S → abB, A → aaBb, B → bbAa, A → λ
- G is context-free and linear, but ~~not regular~~
- L(G)= $\{ab(bbaa)^n bba(ba)^n \mid n \geq 0\}$

$S \Rightarrow abB \Rightarrow abbbAa \Rightarrow abbbaaBba$

$\stackrel{*}{\Rightarrow} ab(bbaa)^2 B (ba)^2$

$\Rightarrow ab(bbaa)^2 bbA a(ba)^2 \Rightarrow ab(bbaa)^2 bba(ba)^2$

Note] Regular and linear grammars are context-free, but a context-free grammar is not necessarily linear.

→ A→AB 이렇게 나올 수도 있다.

Examples of context-free languages(3/3)

Regular (x) Context free (o)

Ex5.3] $L=\{a^nb^m \mid n\neq m\}$ is context free.

$a^n \mid b^n$ $S\to aSb \mid \lambda$

Proof) Produce a context-free grammar for L.

n≥m.
$S_1\to aSb \mid \lambda$
$S\to AS_1$
$A\to aA \mid a$

n<m.
$S_1\to aSb \mid \lambda$
$S\to S_1B$
$B\to bB \mid b$

⟹
$S\to AS_1 \mid S_1B$
$S_1\to aS_1b \mid \lambda$
$A\to aA \mid a$ $B\to bB \mid b$

→ context free grammar

linear x.

$S\to aSb \mid A \mid B$
$A\to aA \mid a$
$B\to bB \mid b$
(linear 하고
context free 하다.)

Ex5.4] G with productions

- $S \to aSb \mid SS \mid \lambda$

aabb

~~ba~~ ~~bba~~
abab.

but not linear ⟹ linear로 만들 수는 있음.

ababaabb

- G is context-free but not linear
- L(G)= $\{w\in\{a,b\}^* \mid n_a(w)=n_b(w),\ n_a(r)\geq n_b(r)$ for any prefix r of $w\}$.
- Related programming languages :
  - Consider homomorphism h(a)= (, h(b)=) 괄호에 해당 한다.

Context free (o) but not linear.

Note] There are many other equivalent grammars, which is sometimes linear.

$\{a^nb^m \mid n\neq m\}$ 을 위한 linear grammar 를 찾아보자.

Leftmost & rightmost derivations

$w \in L(G)$

- Problem of non-linear CFG
  - Derivations with more than one variables
    - There is a choice in the order of applying productions
  - ex) $S \rightarrow AB$, $A \rightarrow aaB \mid \lambda$ $B \rightarrow bB \mid \lambda$ $S \Rightarrow AB$
  - Removing such irrelevant factors
    - Require a specific order for replace of variables.
- Leftmost derivation →왼쪽거 먼저
  - Leftmost variable in the sentential form is replaced.
- Rightmost derivation →오른쪽거 먼저.
  - Rightmost variable in the sentential form is replaced.

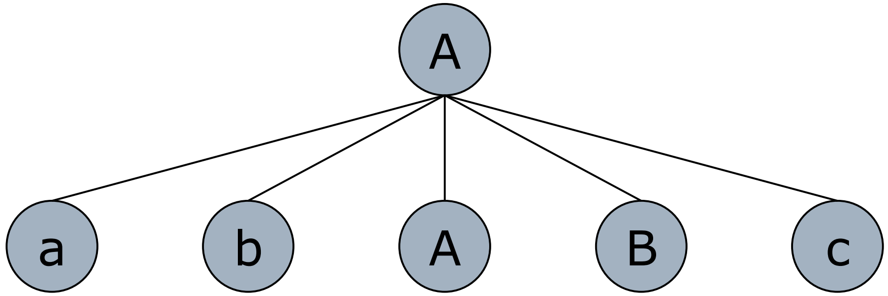
Ex5.5] $S \rightarrow aAB$, $A \rightarrow bBb$, $B \rightarrow A \mid \lambda$

- Leftmost derivation: $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abb.$
- Rightmost derivation: $S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abb.$

미리 정해야 한다.

Derivation Trees (1/3)

- Derivation Trees
  - An ordered tree of showing derivation
    - Nodes : labeled with the left sides of productions
    - Children of a node : its corresponding right sides
    - Root: start symbol
    - Leaves: terminal symbols
  - Independent of the order in which productions are used
  - ex) A → abABc



Derivation Trees (2/3)

[Formal Definition of Derivation Trees]

– An ordered tree is a derivation tree for a CFG $G=(V,T,S,P)$

a vertex v.

⇔ 1. root : labeled S.

2. label of leaf $\in T \cup \{\lambda\}$

3. label of interior 내부 vertex $\in V$.

4. If there exist a vertex with label $A \in V$ and children with labels $a_1, a_2, \ldots, a_n$ (left to right) 순서가 중요하다. then P must contain $A \rightarrow a_1 a_2 \ldots a_n$

5. A leaf labeled λ : has no siblings

$B \rightarrow \bigcirc\,\lambda\,\bigcirc$

이렇게 하면 안된다.

[Partial derivation tree]

A tree with properties 3,4,5 and

2a. label of leaf $\in V \cup T \cup \{\lambda\}$ instead of 2.

[Yield of a tree]

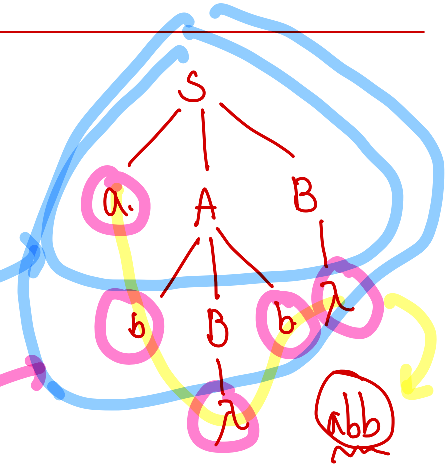
String obtained by reading leaves of the tree from left to right, omitting any λ's encountered.

Derivation Trees (3/3)

Ex5.6] Grammar G with productions

- S → aAB, A → bBb, B → A | λ
- A partial derivation tree sentence | 개념.

의 yield → abBb ≡ sentential form.



abb

- Yield of the tree
- A derivation tree
- Yield of the tree

[note] A derivation tree corresponds to a string

Relation between sentential forms & derivation trees

[Theorem 5.1] Derivation trees & a CFG G=(V,T,S,P)

i) $\forall w \in L(G)$, $\exists$a derivation tree such that its yield is w

ii) The yield of any derivation tree for G is in L(G).

Vertex가 표현된 것.

iii) $\forall$PDT t_G for G with root S, its yield is a sentential form of G

Sketch of Proof)

1. Show that every sentential form of L(G) has a corresponding PDT t_G.
   (Use induction with the number of derivation steps (n))

[Base] Every sentential form (with n=1) has a PDT
[Assumption] Assume that every sentential form (with n=k) has a PDT.
[Inductive step] For every sentential from (with n=k+1),
    we can find a PDT from [Base] and [Assumption].

2. Show that every PDT represents some sentential form.
   (Use Induction with the height (h)of PDT)

3. This can also be applied to derivation trees.

Parsing and Membership(1/6)

L is regular

"$w \overset{?}{\in} L(G)$"

DFA → accept, → reject.

w

- Membership algorithm for L(G)
  - Determine whether w∈L(G) is true or not.
  - For G, need to find a derivation of w

일종의 membership algorithm

$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abb$

CFL → CFG

멤버쉽 알고리즘

- Parsing
  - finding a sequence of productions by which w is derived.

string

L(G) (어떻게 derivate 한지).

  - A membership algorithm can be implemented by parsing
- How to do parsing
  - Systematically construct (all possible derivations) ① and see whether any of them match w.
  - Exhaustive search parsing (top-down parsing)
    1. Looking at all productions of the form S → x
    2. If none of these results in a match with w, apply all applicable productions to the leftmost variable of x
    3. Do this process until we derive w.

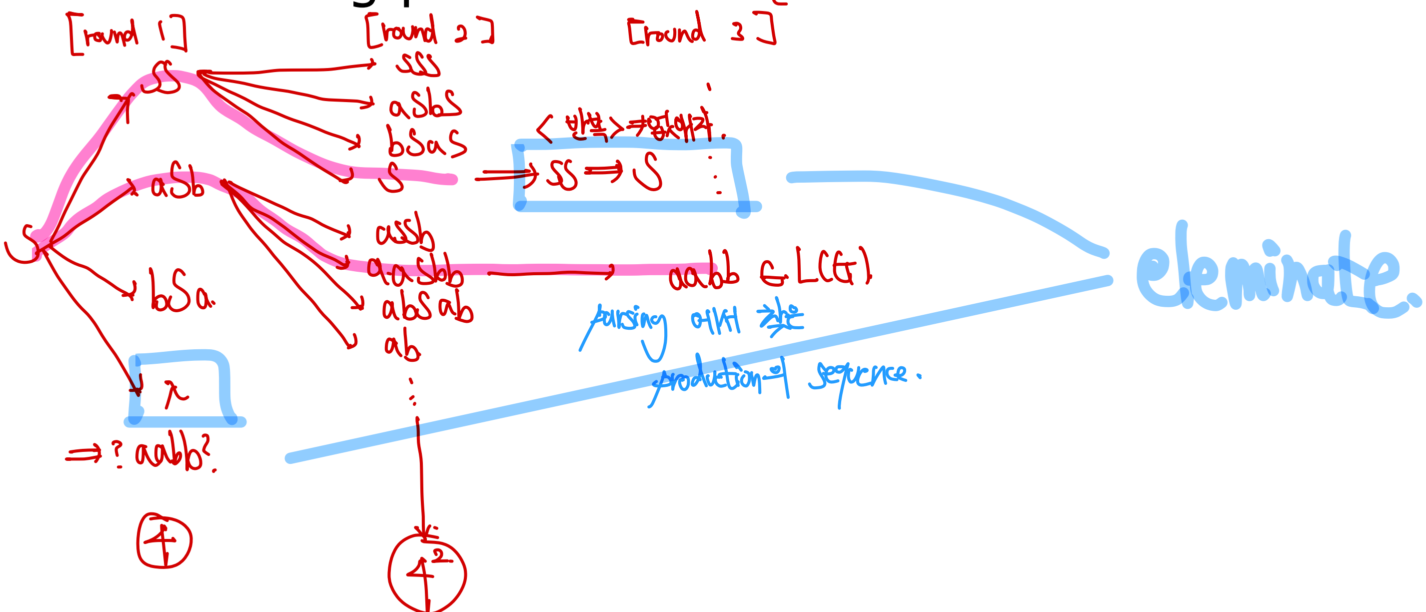
rightmost로 해도 상관x.

Parsing and Membership(2/6)

Ex5.7] S → SS|aSb|bSa|λ.

(Do this process until we derive w.).

– Parsing process of w=aabb



– From this, we can conclude that aabb is in the language.

□ Flaws of exhaustive search parsing

– Tediousness

– Possibility of nontermination for (w ∉ L). 만약 못찾는다면 언제 끝내라는 것이 없다.

Parsing and Membership(3/6)

- A resolution: eliminate production rules of the forms,
  A → λ, A → B

S → B
B → S
S ⇒ B ⇒ S ⇒ B ....

Ex5.8] S → SS|aSb|bSa|ab|ba

- At each round, the length of sentential forms increases.
- Exhaustive search parsing terminate in m rounds (m ≤ |w|)

[round 1]
S → SS, aSb, bSa, ab, ba.
길이 ≥ 2

[round 2]
SS → SSS, aSbS
길이≥3

[round 3]
x

(aaa가 없어.)
그러면 round 3을 볼 필요도 없어
why round 3의 |w|≥4 이기 때문.

(aaa ∈ L(G)) (?)
|aaa| = 3
우리 round 2 에서는 나와야 한다.

sentential form의 길이 > |w|
이면 stop

Parsing and Membership(4/6)

[Theorem 5.2]

없으면.

Consider a CFG G=(V,T,S,P) without A → λ and A → B (A,B∈V.)

For any w ∈Σ*, the exhaustive search parsing can

either produce a parsing of w, $w \in L(G)$

or tell that no parsing is possible. $w \notin L(G)$

$|w| = |aaa| = 3$

Proof) Note the following facts.

이거 때문.

- Each derivation :
  increase the length of sentential form and/or the number of terminal symbols.
- the length of sentential form ≤ |w|
- the number of terminal symbols ≤ |w|

$S \overset{3}{\underset{*}{\Rightarrow}} ABC \overset{3}{\underset{*}{\Rightarrow}} aaa$

Thus, Number of rounds of a derivation ≤ 2|w|.

$S \to SS \to aS \to ab$

최대.

2|w| 초과는 생각할 필요x.

Parsing and Membership(5/6)

Round의 maximum $2|w|$

$\boxed{|w|}$

- How many steps for parsing w?
  - 이론적인 방법일 뿐.
  - [Exhaustive search parsing] for a grammar w/o $A \to \lambda$, $A \to B$ production 의 숫자.
    - Number of steps < $|P|+|P|^2+\ldots+|p|^{2|w|}$ $|p| + |p|^2 + \cdots + |p|^{2|w|}$
  - For every context-free grammars
    - There exist an algorithm parsing w in n steps ($n \propto |w|^3$) 쓸 수 없는 Algorithm
  - Need more efficient parsing! ($n \propto |w|$)

    ↳ 더 나은 Algorithm을 찾아야 한다.

- Simple –grammar (s-grammar)
  - A restricted type of context-free grammar $G=(V,T,S,P)$
  - All productions are of the form,
    - 원래 CFG와 같음. $A \to ax$ where $A \in V$, $a \in T$, $x \in V^*$ Terminal 1개만. Variable 들만 가능.
    - Any pair of (A, a) occurs at most ones in P. Content-free. 일단 linear x

Ex 5.9] $S \to aS \mid bSS \mid c$ s-grammar (o)

(A,a) (A,b) (A,c)

$S \to aS \mid bSS \mid aSS \mid c$. (A,a) 가 중복 ⟹ s-grammar (x)

Parsing and Membership(6/6)

$A_1 \to a_2 B_1 B_2 \cdots B_m$.

- Let G : an s-grammar,

then any $w \in L(G)$ can be parsed with n step, $n \propto |w|$.

Verification)

$S \Rightarrow a_1 A_1 A_2 \cdots A_m \Rightarrow a_1 a_2 B_1 B_2 \cdots B_m$.

(S, a_1)

consider a string $w = a_1 a_2 \ldots a_n$.

$S \to a_1 A_1 A_2 \cdots A_m$

Start of derivation : $S \Rightarrow a_1 A_1 A_2 \ldots A_m$,

Substitution of A1 : there is a unique choice

$$S \Rightarrow a_1 a_2 B_1 B_2 \ldots A_2 \ldots A_m$$

Like these, each step produces one terminal symbol.

Thus, the whole process must be completed in no more than |w| steps.

|2w| (x) 무관. 참.

sentential form의 길이 ↑

s-grammar

(또 Context free. s-grammar X).

Ex5.9] parsing of aabbccc with $S \to aS|bSS|c$

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aabSS \Rightarrow aabbSSS \Rightarrow aabbcSS \Rightarrow aabbccS \Rightarrow \boxed{aabbccc}$.

애매모호하다

Ambiguity in Grammars and Languages(1/3)

[Definition] Ambiguity of a sentence w

- A number of different derivation tree may exist
- Two or more leftmost or rightmost derivations exist

[Definition] Ambiguity of a grammar G

A CFG G is said to be ambiguous

when There exists some $w \in L(G)$ with ambiguity

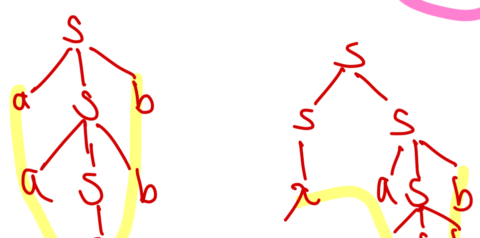
Ex5.10] G with productions $S \rightarrow aSb|SS|\lambda$ is ambiguous.

→ ambiguous 함을 보여라.

= ambiguous 한 w.

- Consider the sentence aabb

→ ambiguous 한지 how



G is ambiguous

□ When defining a programming language,

- required to remove the ambiguity
- by rewriting the grammar in an unambiguous form.

Ambiguity in Grammars and Languages(2/3)

Ex5.11] G=(V,T,E,P), V={E,I}, T={a,b,c,+,*,(,)}

E → I | E+E | E*E | (E), I → a|b|c

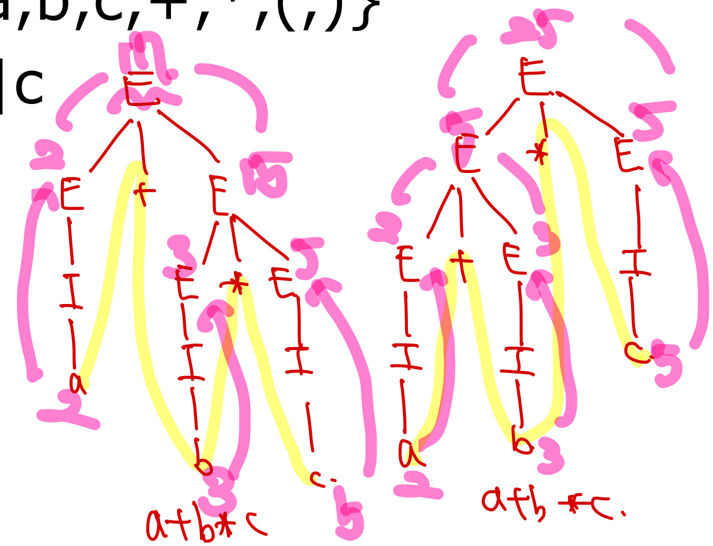
- The grammar G is ambiguous.

X

없애야 한다.

- Consider a+b*c

2. 3 5.



- Way to resolve the ambiguity

  : to associate precedence rules with the operators + and *.

  : to rewrite the grammar.

unambi.

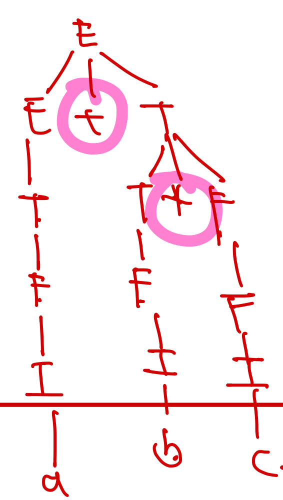
=.

원래 순서를

바꿀 수 없도록

Ex5.12] Rewriting of G in Ex5.11

- Introduce new variables taking V={E,T,F,I}
- E → T | E+T, T → F | T*F, F → I | (E), I → a|b|c
- Derivation of a+b*c :



Ambiguity in Grammars and Languages(3/3)

[Definition] Ambiguity of Language

- A context-free language L is **unambiguous** if there exits an unambiguous grammar.
- A Language L is called 본질적으로 "**inherently** **ambiguous**" if every grammar generating L is ambiguous. 모호성 제거해야 할 수 없는. ambiguous

모든 grammar가.

Ex5.12] L={$a^n b^n c^m$} ∪ {$a^n b^m c^m$} (n,m ≥ 0)

= inherently ambiguous.

$\boxed{a+b*c}$ → unambiguous language).

- Make a context-free grammar G generating L

  $S \rightarrow AC \mid DB$
  $A \rightarrow aAb \mid \lambda$
  $C \rightarrow cC \mid \lambda$
  $B \rightarrow bBc \mid \lambda$
  $D \rightarrow aD \mid \lambda$

  ambiguous 하다
  ⇒

- Check the ambiguity of G

  $S \Rightarrow AC \Rightarrow aAbC \Rightarrow abC \Rightarrow abcC \Rightarrow abc$.
  $\Rightarrow DB \Rightarrow DbBc \Rightarrow Dbc \Rightarrow aDbc \Rightarrow abc$.). 두개의 다른.

  $a^n b^n c^n$

  "ambiguous"

- Is L inherently ambiguous? 모든 G를 다. 해봐야 함.

Not clear.

Formal language for Programming Languages

- Definition of Grammar → definition of a programming languages
- Parsing → interpreter & Compiler

  문법에 해당 되는지. derivation.
- Definition of a PL by grammar : BNF (Backus-Naur Form)
  - Ex5.12 revisited] definition using BNF

    $E \to T \mid E+T$

    <expression> ::= <term>|<expression>+<term>

    <term> ::= <factor> | <term> * <factor>
- Example of s-grammar

  <if_statement>::= if<expression><then_clause><else_clause>
- Difficulties
  - Specification of a PL must be unambiguous.
  - It is not easy to deciding inherent ambiguity of a PL.

    쉽지 않다.
  - Some semantic features may be poorly defined or ambiguous.