Bab 5 Context-Free Grammars

Formalism

Derivations

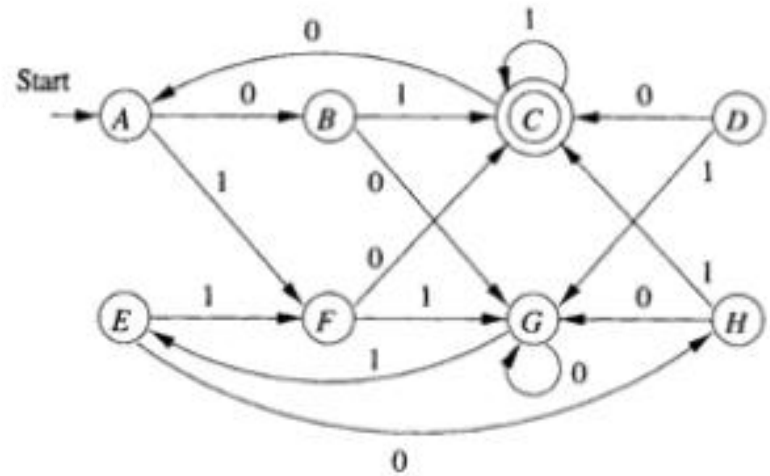
Backus-Naur Form

Left- and Rightmost Derivations

Informatika

Review Minggu Lalu

- **Product Automaton**
  - Equivalence 2 FA (apakah L = M)
  - Containment 2 FA (apakah $L \subseteq M$)
- **Minimisasi State dg Table Filling Algorithm (menggunakan Distinguishable Table)**



	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

- **Buatlah sebuah Language yang dapat menerima string palindrom**
  - Contoh:
    - MALAM
    - MAKAM
    - KASUR NABABAN RUSAK

Informal Comments

- **A *context-free grammar* is a notation for describing languages.**
- **It is more powerful than finite automata or RE's, but still cannot define all possible languages.**
- **Useful for nested structures, e.g., parentheses in programming languages.**

Informal Comments – (2)

- **Basic idea is to use “variables” to stand for sets of strings (i.e., languages).**
- **These variables are defined recursively, in terms of one another.**
- **Recursive rules (“productions”) involve only concatenation.**
- **Alternative rules for a variable allow union.**

Example: CFG for { 0^n1^n | n $\geq$ 1}

- **Productions:**

  S -> 01

  S -> 0S1

- **Basis: 01 is in the language.**

- **Induction: if w is in the language, then so is 0w1.**

CFG Formalism

- ***There are 4 components.***
- ***Terminals*** **= symbols of the alphabet of the language being defined.**
- ***Variables*** **=** ***nonterminals*** **= a finite set of other symbols, each of which represents a language.**
- ***Start symbol*** **= the variable whose language is the one being defined.**

Productions

- **A *production* has the form variable -> string of variables and terminals.**
- **Convention:**
  - A, B, C,… are variables.
  - a, b, c,… are terminals.
  - …, X, Y, Z are either terminals or variables.
  - …, w, x, y, z are strings of terminals only.
  - α, β, γ,… are strings of terminals and/or variables.

Example: Formal CFG

- **Here is a formal CFG for { 0^n1^n | n $\geq$ 1}.**
- **Terminals = {0, 1}.**
- **Variables = {S}.**
- **Start symbol = S.**
- **Productions =**

  S -> 01

  S -> 0S1

Derivations – Intuition

- **We *derive* strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the right side of one of its productions.**
  - That is, the "productions for A" are those that have A on the left side of the ->.

Derivations – Formalism

- We say $\alpha A\beta$ => $\alpha\gamma\beta$ if A -> γ is a production.
- Example: S -> 01; S -> 0S1.
- S => 0S1 => 00S11 => 000111.

Iterated Derivation

- **=>* means "zero or more derivation steps."**
- **Basis: α =>* α for any string α.**
- **Induction: if α =>* β and β => γ, then α =>* γ.**

Example: Iterated Derivation

- **S -> 01; S -> 0S1.**
- **S => 0S1 => 00S11 => 000111.**
- **So, S =>* S; S =>* 0S1; S =>* 00S11; S =>* 000111.**

Sentential Forms

- **Any string of variables and/or terminals derived from the start symbol is called a *sentential form*.**
- **Formally, α is a sentential form iff $S \Rightarrow^* \alpha$.**

Language of a Grammar

- **If G is a CFG, then L(G), the *language of G*, is $\{w \mid S \Rightarrow^* w\}$.**
  - Note: w must be a terminal string, S is the start symbol.
- **Example: G has productions S -> ϵ and S -> 0S1.**
- **$L(G) = \{0^n1^n \mid n \geq 0\}$.**

Note: ϵ is a legitimate right side.

Example: $G_{pal} = (\{P\}, \{0, 1\}, A, P)$, where $A = \{P \to \epsilon, P \to 0, P \to 1, P \to 0P0, P \to 1P1\}$.

Sometimes we group productions with the same head, e.g. $A = \{P \to \epsilon|0|1|0P0|1P1\}$.

Context-Free Languages

- **A language that is defined by some CFG is called a *context-free language*.**
- **There are CFL's that are not regular languages, such as the example just given.**
- **But not all languages are CFL's.**
- **Intuitively: CFL's can count two things, not three.**

Buatlah sebuah CFG yang menerima Bahasa {w | length of w is odd} dengan symbol a dan b

$\{w \mid |w| \text{ is odd}\}$

S→ A |B

A → aSS | a

B → bSS | b

$\{0^n 1^{2n} \mid n>0\}$

S→ 011 | 0S11

$\{0^n 1^n \mid n>0\} \cup \{0^n 1^{2n} \mid n>0\}$

S→ B | A

B → 01 | 0B1

A → 011 | 0A11

$\{ a^n b^m \mid 0 \leq n \leq m \leq 2n \}$

S→ eps

~~S → ab~~

~~S → abb~~

S→ aSb

S→aSbb

Aabbb: S→aSbb→aaSbbb→aabbb

BNF Notation

- **Grammars for programming languages are often written in BNF (*Backus-Naur Form*).**
- **Variables are words in <…>; Example: <statement>.**
- **Terminals are often multicharacter strings indicated by boldface or underline; Example: while or <u>WHILE</u>.**

BNF Notation – (2)

- **Symbol ::= is often used for ->.**
- **Symbol | is used for "or."**
    - A shorthand for a list of productions with the same left side.
- **Example: S -> 0S1 | 01 is shorthand for S -> 0S1 and S -> 01.**

BNF Notation – Kleene Closure

- **Symbol … is used for "one or more."**
- **Example: <digit> ::= 0|1|2|3|4|5|6|7|8|9**

<unsigned integer> ::= <digit>…

  - Note: that's not exactly the * of RE's.

- **Translation: Replace α… with a new variable A and productions A -> Aα | α.**

Example: Kleene Closure

- **Grammar for unsigned integers can be replaced by:**

  U -> UD | D

  D -> 0|1|2|3|4|5|6|7|8|9

BNF Notation: Optional Elements

- **Surround one or more symbols by [...] to make them optional.**
- **Example: <statement> ::= if <condition> then <statement> [; else <statement>]**
- **Translation: replace [α] by a new variable A with productions A -> α | ϵ.**

Example: Optional Elements

- **Grammar for if-then-else can be replaced by:**

S -> iCtSA

A -> ;eS | ϵ

BNF Notation – Grouping

- **Use {…} to surround a sequence of symbols that need to be treated as a unit.**
  - Typically, they are followed by a … for “one or more.”
- **Example: <statement list> ::= <statement> [{;<statement>}…]**

Translation: Grouping

- **You may, if you wish, create a new variable A for {α}.**
- **One production for A: A -> α.**
- **Use A in place of {α}.**

Example: Grouping

L -> S [{;S}…]

- **Replace by L -> S [A…]   A -> ;S**
  - A stands for {;S}.
- **Then by L -> SB   B -> A… | ϵ   A -> ;S**
  - B stands for [A…] (zero or more A's).
- **Finally by L -> SB   B -> C | ϵ   C -> AC | A   A -> ;S**
  - C stands for A… .

Leftmost and Rightmost Derivations

- Derivations allow us to replace any of the variables in a string.
- Leads to many different derivations of the same string.
- By forcing the leftmost variable (or alternatively, the rightmost variable) to be replaced, we avoid these "distinctions without a difference."

Leftmost Derivations

- **Say $wA\alpha \Rightarrow_{lm} w\beta\alpha$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.**
- **Also, $\alpha \Rightarrow^*_{lm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{lm}$ steps.**

Example: Leftmost Derivations

- **Balanced-parentheses grammmar:**

  S -> SS | (S) | ()

- $S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$
- **Thus,** $S \Rightarrow^*_{lm} (())()$
- $S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$ **is a derivation, but not a leftmost derivation.**

Rightmost Derivations

- **Say $\alpha Aw \Rightarrow_{rm} \alpha\beta w$ if w is a string of terminals only and A -> β is a production.**
- **Also, $\alpha \Rightarrow^*_{rm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{rm}$ steps.**

Example: Rightmost Derivations

- **Balanced-parentheses grammmar:**

  S -> SS | (S) | ()

- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- **Thus,** $S \Rightarrow^*_{rm} (())()$
- **S => SS => SSS => S()S => ()()S => ()()() is neither a rightmost nor a leftmost derivation.**

Parse Trees

Definitions

Relationship to Left- and Rightmost Derivations

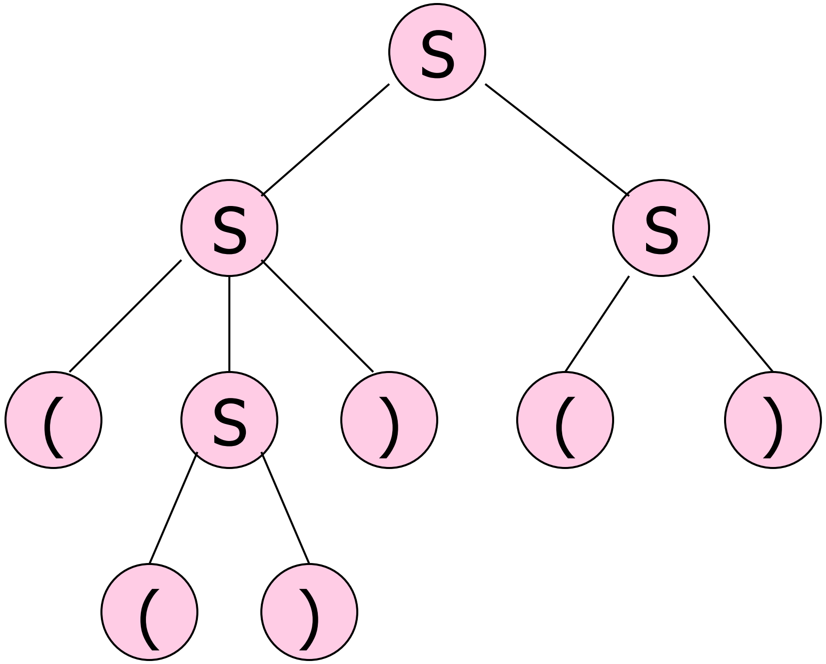
Ambiguity in Grammars

Parse Trees

- ***Parse trees*** **are trees labeled by symbols of a particular CFG.**
- **Leaves: labeled by a terminal or ϵ.**
- **Interior nodes: labeled by a variable.**
  - Children are labeled by the right side of a production for the parent.
- **Root: must be labeled by the start symbol.**

Example: Parse Tree for (())()

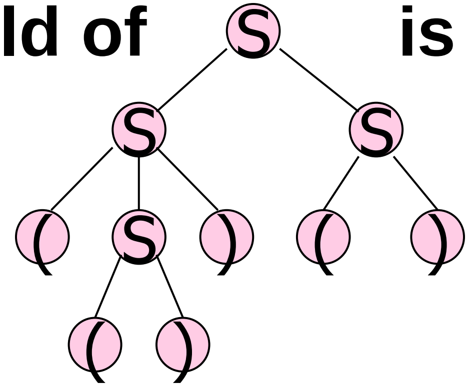
S -> SS | (S) | ()



Yield of a Parse Tree

- **The concatenation of the labels of the leaves in left-to-right order**
  - That is, in the order of a preorder traversal.

is called the *yield* of the parse tree.

- **Example: yield of**  **is (())()**

Parse Trees, Left- and Rightmost Derivations

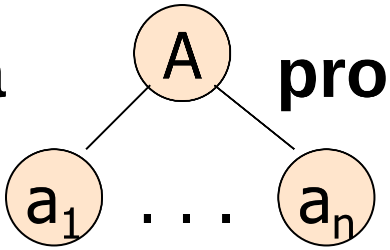
- **For every parse tree, there is a unique leftmost, and a unique rightmost derivation.**

- **We'll prove:**

  1. If there is a parse tree with root labeled A and yield w, then $A \Rightarrow^*_{lm} w$.
  2. If $A \Rightarrow^*_{lm} w$, then there is a parse tree with root A and yield w.

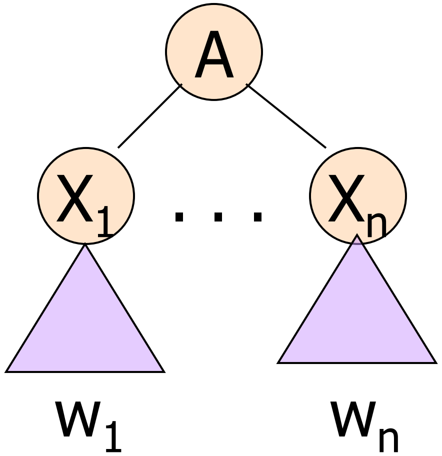
Proof – Part 1

- **Induction on the *height* (length of the longest path from the root) of the tree.**
- **Basis: height 1. Tree looks like**
- **$A \to a_1 \ldots a_n$ must be a production.**
- **Thus, $A \Rightarrow^*_{lm} a_1 \ldots a_n$.**



Part 1 – Induction

- **Assume (1) for trees of height < h, and let this tree have height h:**
- **By IH, $X_i \Rightarrow^*_{lm} w_i$.**
  - Note: if X_i is a terminal, then $X_i = w_i$.
- **Thus, $A \Rightarrow_{lm} X_1 \ldots X_n \Rightarrow^*_{lm} w_1X_2 \ldots X_n \Rightarrow^*_{lm} w_1w_2X_3 \ldots X_n \Rightarrow^*_{lm} \ldots \Rightarrow^*_{lm} w_1 \ldots w_n$.**

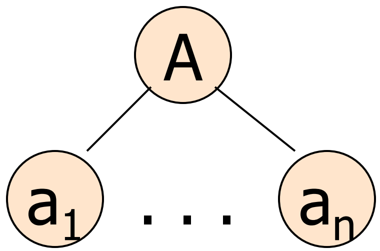


Proof: Part 2

- Given a leftmost derivation of a terminal string, we need to prove the existence of a parse tree.
- The proof is an induction on the length of the derivation.

Part 2 – Basis

- If $A \Rightarrow^*_{lm} a_1 \ldots a_n$ by a one-step derivation, then there must be a parse tree

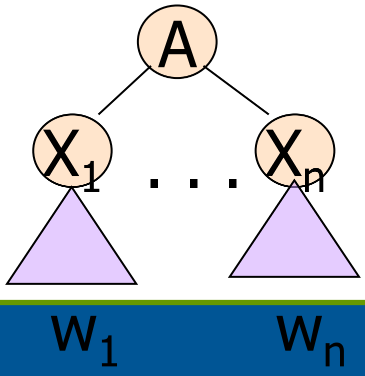


Part 2 – Induction

- **Assume (2) for derivations of fewer than k > 1 steps, and let $A \Rightarrow^*_{lm} w$ be a k-step derivation.**
- **First step is $A \Rightarrow_{lm} X_1 \ldots X_n$.**
- **Key point: w can be divided so the first portion is derived from X_1, the next is derived from X_2, and so on.**
  - If X_i is a terminal, then $w_i = X_i$.

Induction – (2)

- **That is, X_i =>*$_{lm}$ w_i for all i such that X_i is a variable.**
  - And the derivation takes fewer than k steps.
- **By the IH, if X_i is a variable, then there is a parse tree with root X_i and yield w_i.**
- **Thus, there is a parse tree**



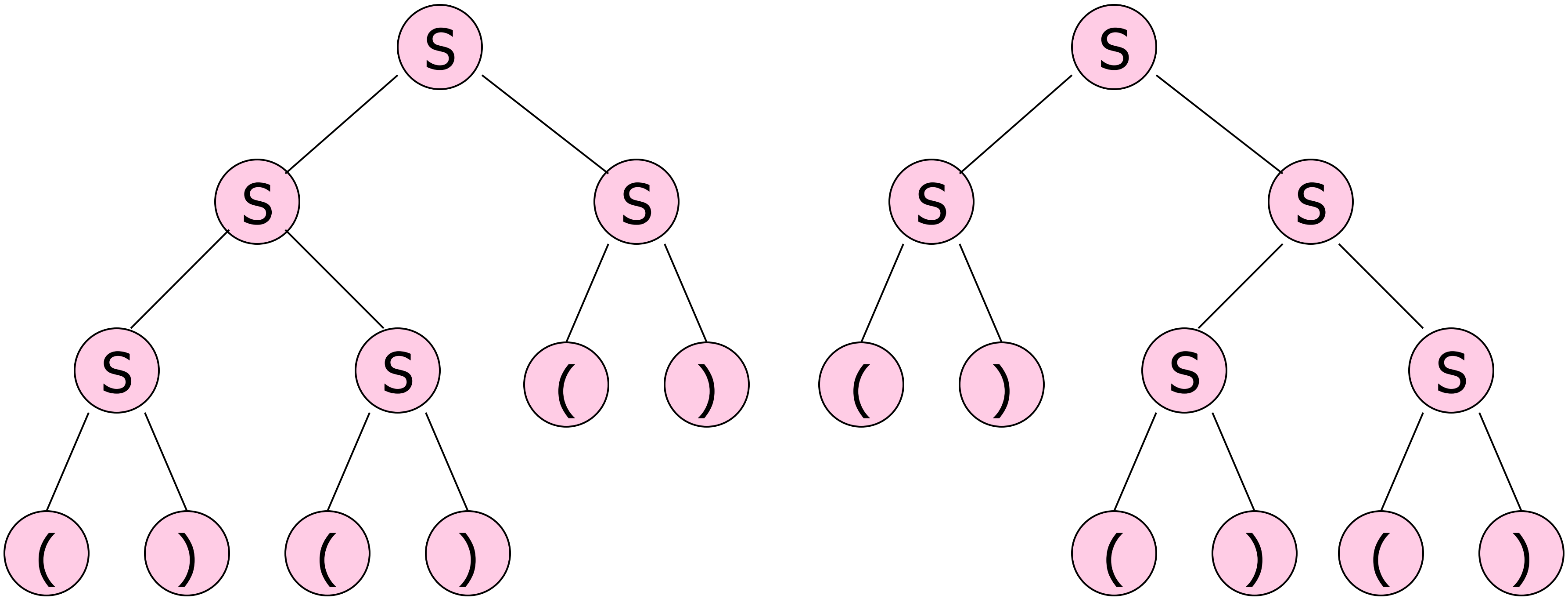
Parse Trees and Rightmost Derivations

- The ideas are essentially the mirror image of the proof for leftmost derivations.
- Left to the imagination.

Ambiguous Grammars

- **A CFG is *ambiguous* if there is a string in the language that is the yield of two or more parse trees.**
- **Example: S -> SS | (S) | ()**
- **Two parse trees for ()()() on next slide.**

Example – Continued



Example: (simple) expressions in a typical prog lang. Operators are $+$ and $*$, and arguments are identfiers, i.e. strings in
$L((\mathbf{a}+\mathbf{b})(\mathbf{a}+\mathbf{b}+\mathbf{0}+\mathbf{1})^*)$

The expressions are defined by the grammar

$$G = (\{E, I\}, T, P, E)$$

where $T = \{+, *, (,), a, b, 0, 1\}$ and P is the following set of productions:

1. $E \longrightarrow I$
2. $E \longrightarrow E + E$
3. $E \longrightarrow E * E$
4. $E \longrightarrow (E)$
5. $I \longrightarrow a$
6. $I \longrightarrow b$
7. $I \longrightarrow Ia$
8. $I \longrightarrow Ib$
9. $I \longrightarrow I0$
10. $I \longrightarrow I1$

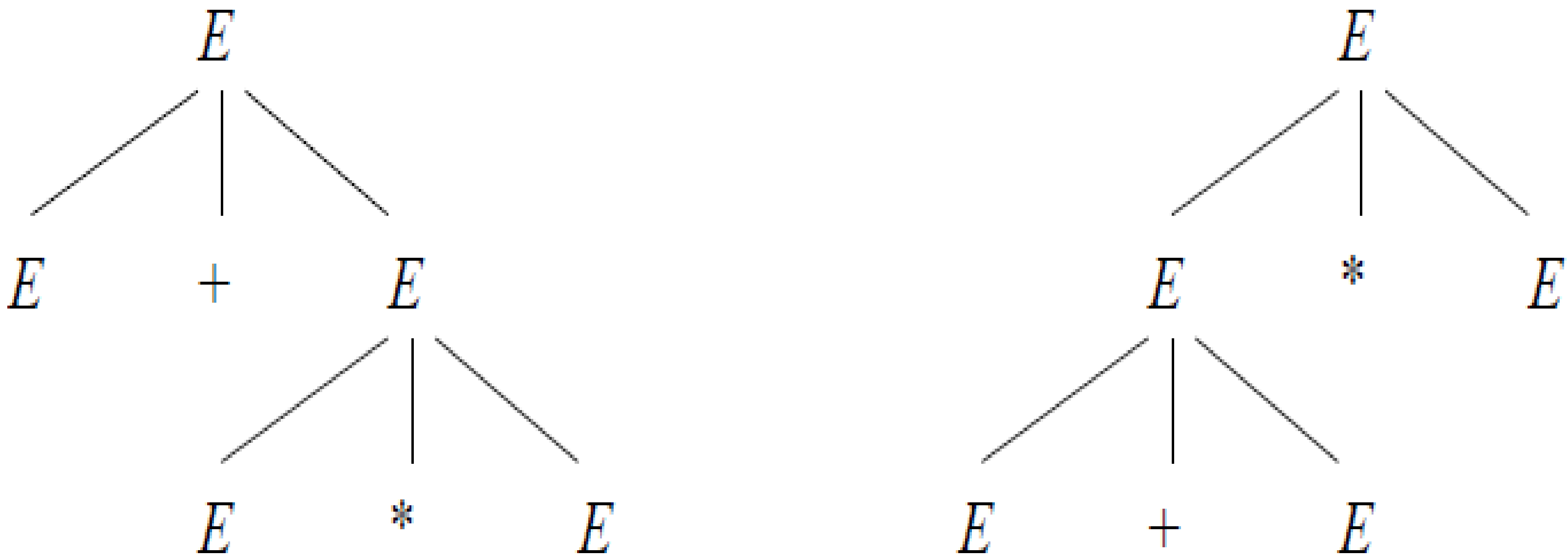
the sentential form $E + E * E$ has two derivations:

$$E \Rightarrow E + E \Rightarrow E + E * E$$

and

$$E \Rightarrow E * E \Rightarrow E + E * E$$

This gives us two parse trees:



(a) (b)

Ambiguity, Left- and Rightmost Derivations

- **If there are two different parse trees, they must produce two different leftmost derivations by the construction given in the proof.**
- **Conversely, two different leftmost derivations produce different parse trees by the other part of the proof.**
- **Likewise for rightmost derivations.**

Ambiguity, etc. – (2)

- **Thus, equivalent definitions of "ambiguous grammar" are:**
  1. There is a string in the language that has two different leftmost derivations.
  2. There is a string in the language that has two different rightmost derivations.

Ambiguity is a Property of Grammars, not Languages

- **For the balanced-parentheses language, here is another CFG, which is unambiguous.**

B, the start symbol, derives balanced strings.

B -> (RB | ϵ

R ->) | (RR

R generates strings that have one more right paren than left.

Example: Unambiguous Grammar

B -> (RB | ϵ   R ->) | (RR

- **Construct a unique leftmost derivation for a given balanced string of parentheses by scanning the string from left to right.**
  - If we need to expand B, then use B -> (RB if the next symbol is "(" and ϵ if at the end.
  - If we need to expand R, use R ->) if the next symbol is ")" and (RR if it is "(".

The Parsing Process

Remaining Input:

(())()

Next symbol

Steps of leftmost derivation:

B

B -> (RB | ϵ R ->) | (RR

The Parsing Process

Remaining Input:

())()

Next symbol

Steps of leftmost derivation:

B

(RB

B -> (RB | ε   R ->) | (RR

The Parsing Process

Remaining Input:

))()

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

)()

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

B -> (RB | ϵ R ->) | (RR

The Parsing Process

Remaining Input:

()

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

B -> (RB | ϵ R ->) | (RR

The Parsing Process

Remaining Input:

)

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

(())(RB

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Steps of leftmost derivation:

Remaining Input:

Next symbol

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Steps of leftmost derivation:

Remaining Input:

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

(())()

Next symbol

B -> (RB | ϵ   R ->) | (RR

LL(1) Grammars

- **As an aside, a grammar such B -> (RB | ϵ R ->) | (RR, where you can always figure out the production to use in a leftmost derivation by scanning the given string left-to-right and looking only at the next one symbol is called LL(1).**
  - “Leftmost derivation, left-to-right scan, one symbol of lookahead.”

LL(1) Grammars – (2)

- **Most programming languages have LL(1) grammars.**
- **LL(1) grammars are never ambiguous.**

Inherent Ambiguity

- **It would be nice if for every ambiguous grammar, there were some way to "fix" the ambiguity, as we did for the balanced-parentheses grammar.**
- **Unfortunately, certain CFL's are *inherently ambiguous*, meaning that every grammar for the language is ambiguous.**

Example: Inherent Ambiguity

- **The language $\{0^i1^j2^k \mid i = j \text{ or } j = k\}$ is inherently ambiguous.**
- **Intuitively, at least some of the strings of the form $0^n1^n2^n$ must be generated by two different parse trees, one based on checking the 0's and 1's, the other based on checking the 1's and 2's.**

One Possible Ambiguous Grammar

S -> AB | CD

A -> 0A1 | 01

B -> 2B | 2

C -> 0C | 0

D -> 1D2 | 12

A generates equal 0's and 1's

B generates any number of 2's

C generates any number of 0's

D generates equal 1's and 2's

And there are two derivations of every string with equal numbers of 0's, 1's, and 2's. E.g.:

S => AB => 01B => 012

S => CD => 0D => 012

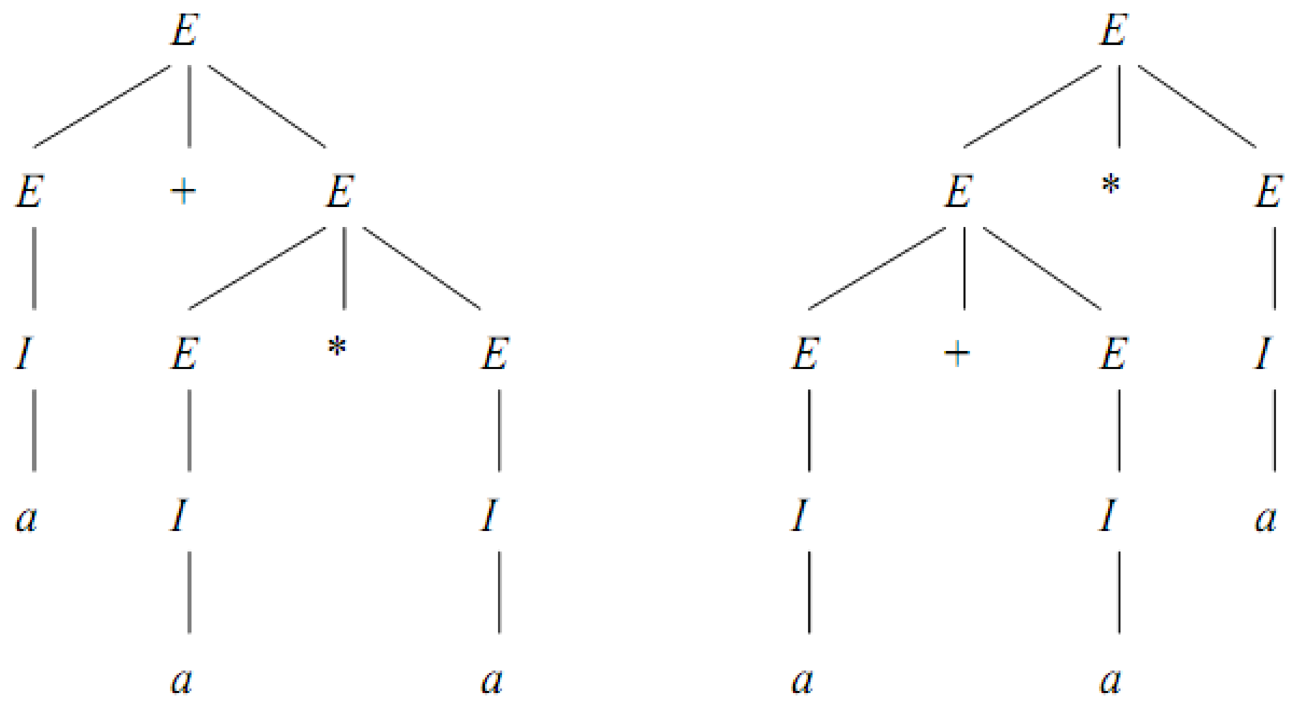
The expressions are defined by the grammar

$$G = (\{E, I\}, T, P, E)$$

where $T = \{+, *, (,), a, b, 0, 1\}$ and P is the following set of productions:

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Example: The terminal string $a + a * a$ has two parse trees:



(a) (b)

Latihan

Desain context free grammar :

- **menerima string () yg berpasangan, contoh: (), (()), ()()**
- **menerima string berupa ekspresi matematika untuk simbol terminal {0, 1, +, *, (,)}.**
- **menerima string dengan jumlah 1 adalah dua kali jumlah 0**

{w | w starts and ends with the same symbol}

$$\{a^n b^m c^k : k = n + m\}$$

$\{0^i 1^j 2^k \mid \quad i=j \text{ or } j=k\}$

$\{0^i 1^j 2^k \mid i \neq j \text{ or } j \neq k\}$

$\{w\#x \mid w^R \text{ is a substring of } x, \text{ where } w, x \in \{a, b\}^*\}$

$\{w\#x \mid w^R \text{ is a substring of } x, \text{ where } w, x \in \{a, b\}^*\}$