DFA NFA RE

RL CFL → CFG

Context-Free Grammars

Formalism

Derivations

Backus-Naur Form

Left- and Rightmost Derivations

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)

Non - Terminal

- 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

A → 0

B → A1

Terminal (แทนไม่ได้) = 0 1

Non-Terminal = S

(แทนได้อยู่)

Start Symbol = ตัวหลักสุดของกฎ

- Basis: 01 is in the language.
- Induction: if w is in the language, then so is 0w1.

CFG Formalism

- *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. NT
  - a, b, c,… are terminals. T
  - …, 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

000111

3 3

- Here is a formal CFG for $\{ 0^n1^n \mid n \geq 1\}$.
- Terminals = {0, 1}.

S => 0S1

- Variables = {S}.

=> 00S11

=> 000111

- Start symbol = S.
- Productions =

  S -> 01

  S -> 0S1

การแทนที่

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

Iterated Derivation

- $=>*$ means "zero or more derivation steps."
- Basis: $\alpha =>* \alpha$ for any string α.
- Induction: if $\alpha =>* \beta$ and $\beta => \gamma$, then $\alpha =>* \gamma$.

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*. เอาหมด.
- Formally, α is a sentential form iff S =>* α.

Language of a Grammar

- If G is a CFG, then L(G), the *language of G*, is {w | S =>* 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.

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.

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>…

< > = Loop

  - Note: that's not exactly the * of RE's.
- Translation: Replace α… with a new variable A and productions A -> $A\alpha$ | α.

ตัวหยุด

Recursive

เรียกตัวเองไปๆ

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:

if

then

S -> iCtSA

A -> ;eS | ϵ

condition

else

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 $\{\alpha\}$.
- One production for A: A -> α.
- Use A in place of $\{\alpha\}$.

Example: Grouping

L -> S [{;S}…]

- Replace by L -> S [A…]   A -> ;S   สร้างกฎ
  - 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… .

A ตัวเดียว

A มากกว่า 1 ครั้ง

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 => SS => S() => (S)() => (())() is a derivation, but not a leftmost derivation.

Rightmost Derivations

- Say $\alpha A w \Rightarrow_{rm} \alpha\beta w$ 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

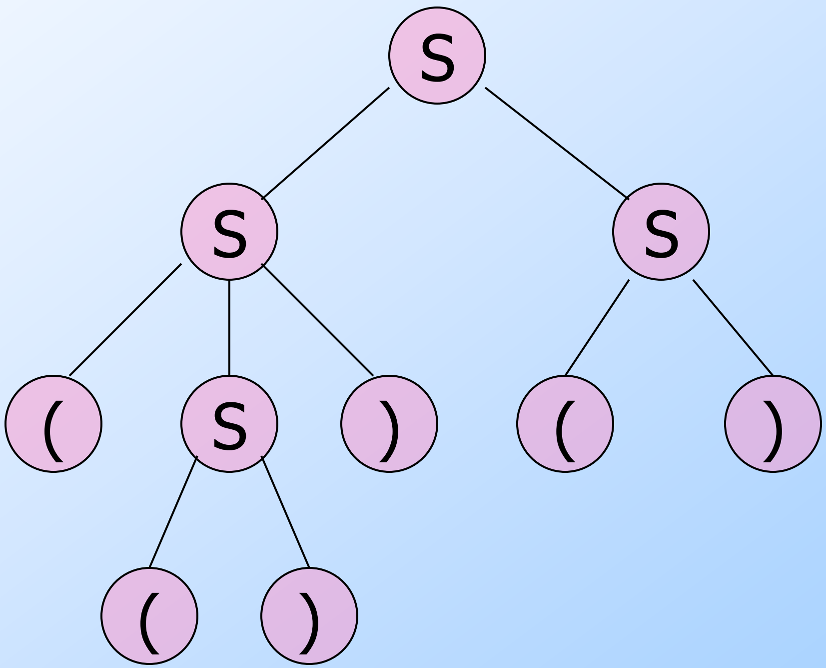
NT = { A, B }
T = { a, b, ε }

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.

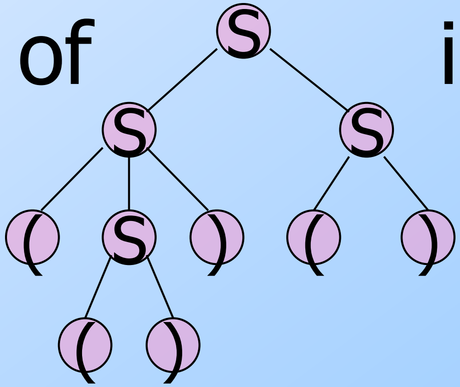
Example: Parse Tree

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 $A =>^*_{lm} w$.
  2. If $A =>^*_{lm} w$, then there is a parse tree with root A and yield w.

Code
หลายตัว => meaning
เดียว

Ambiguous Grammars

ไม่กำกวม.
หรือ code เดียวกัน
ความหมาย ต้องเหมือนกัน

- A CFG is *ambiguous* if there is a string in the language that is the yield of two or more parse trees.

ห้ามมีหลาย Meaning

- Example: S -> SS | (S) | ()
- Two parse trees for ()()() on next slide.

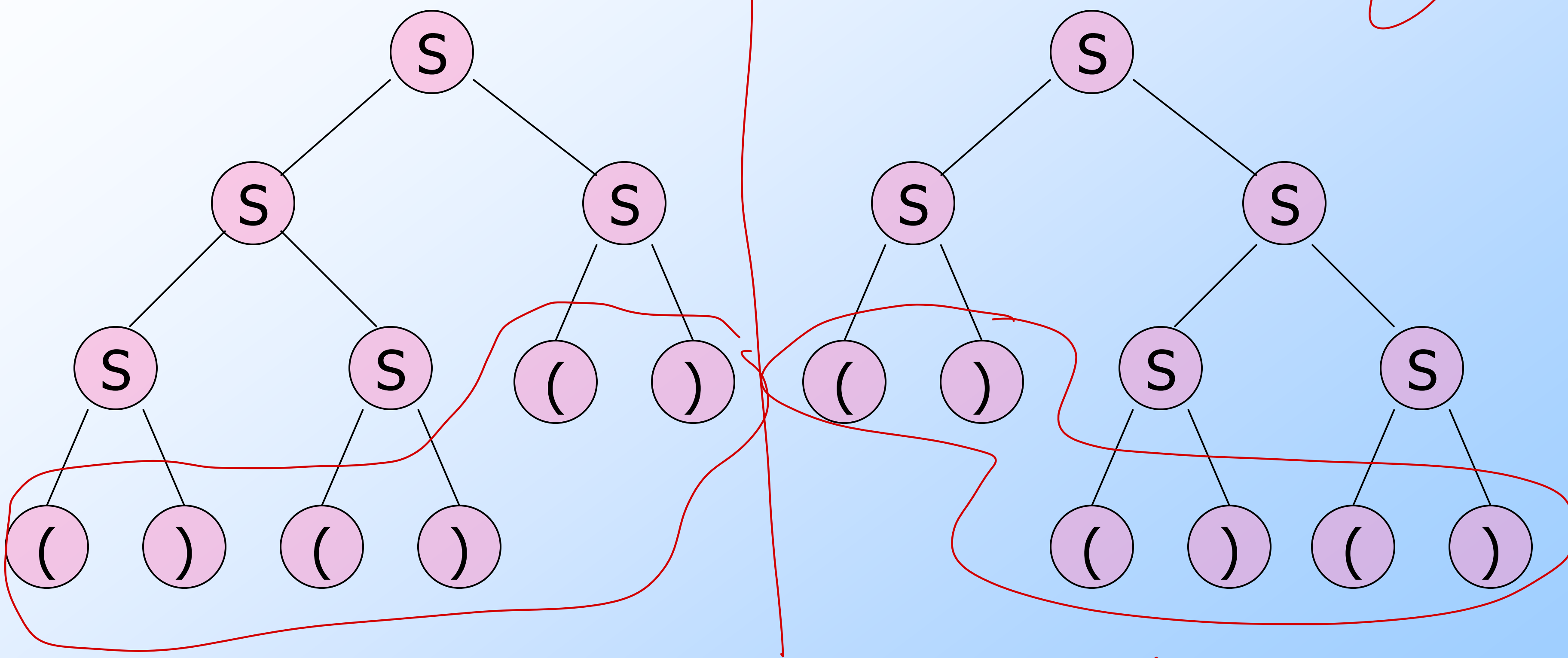
S → SS | (S) | ( )

Parse tree ได้มากกว่า 1 แบบ
กำกวม.

Example – Continued

1

2



( ) ( ) ( )

( ) ( ) ( )

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

ครบคู่ วงเล็บ

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

B -> (RB | ε

R ->) | (RR

B, the start symbol, derives balanced strings.

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

NT: {B, R}

T: {(,), ε}

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

Remaining Input:

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

(())(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
(())()

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 กำกวม. 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.

ได้ 2 อัน Parse Tree

One Possible Ambiguous Grammar

ไม่ตรง check

S -> AB | CD

A -> 0A1 | 01 — A generates equal 0's and 1's

B -> 2B | 2 — B generates any number of 2's

C -> 0C | 0 — C generates any number of 0's

D -> 1D2 | 12 — 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