CS5371
Theory of Computation

Lecture 7: Automata Theory V
(CFG, CFL, CNF)

Objectives

- Introduce Context-free Grammar (CFG) and Context-free Language (CFL)
- Show that Regular Language can be described by CFG
- Terminology related to CFG
  - Leftmost Derivation, Ambiguity, Chomsky Normal Form (CNF)
- Converting a CFG into CNF

Context-free Grammar
(Example)

Substitution Rules

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

Variables	A, B
Terminals	0,1,#
Start Variable	A

Important: Substitution Rule in CFG has a special form:

Exactly one variable (and nothing else) on the left side of the arrow

How does CFG generate strings?

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

- Write down the start symbol
- Find a variable that is written down, and a rule that starts with that variable; Then, replace the variable with the rule
- Repeat the above step until no variable is left

How does CFG generate strings?

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

Step 1. A (write down the start variable)

Step 2. $0A1$ (find a rule and replace)

Step 3. $00A11$ (find a rule and replace)

Step 4. $00B11$ (find a rule and replace)

Step 5. $00\#11$ (find a rule and replace)

Now, the string 00#11 does not have any variable.
We can stop.

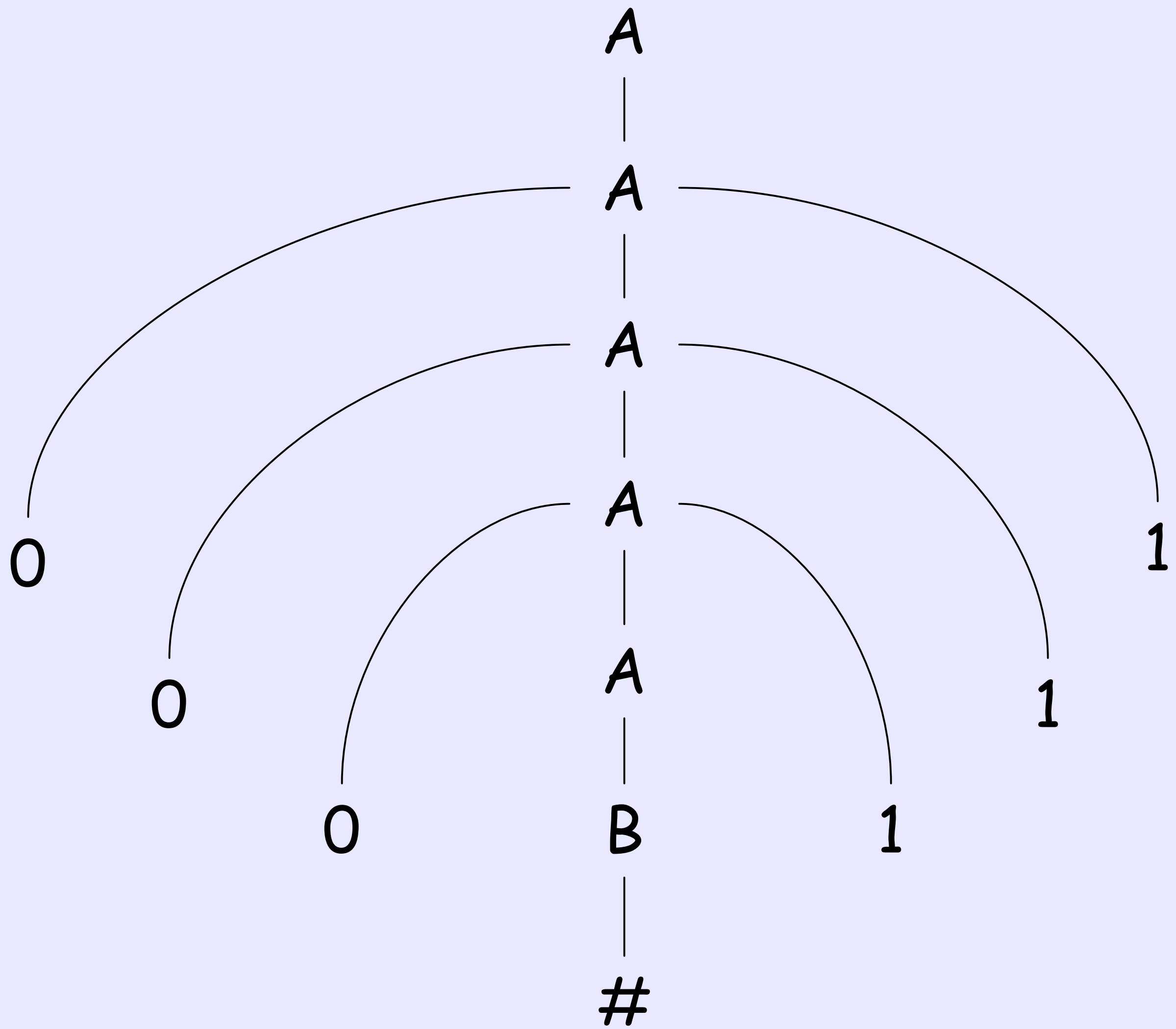
How does CFG generate strings?

- The sequence of substitutions to generate a string is called a derivation
- E.g., A derivation of the string 000#111 in the previous grammar is

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111$$
$$\Rightarrow 000B111 \Rightarrow 000\#111$$

- The same information can be represented pictorially by a parse tree (next slide)

Parse Tree



Language of the Grammar

- In fact, the previous grammar can generate strings #, 0#1, 00#11, 000#111, …
- The set of all strings that can be generated by a grammar G is called the language of G, denoted by L(G)
- The language of the previous grammar is $\{0^n\#1^n \mid n \geq 0\}$

CFG (Formal Definition)

- A CFG is a 4-tuple (V,T, R, S), where
  - V is a finite set of variables
  - T is a finite set of terminals
  - R is a set of substitution rules, where each rule consists of a variable (left side of the arrow) and a string of variables and terminals (right side of the arrow)
  - $S \in V$ called the start variable

CFG (terminology)

- Let u and v be strings of variables and terminals
- We say u derives v, denoted by $u \overset{*}{\Rightarrow} v$, if
  - $u = v$, or
  - there exists $u_1, u_2, \ldots, u_k$, $k \geq 0$ such that $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$
- In other words, for a grammar $G = (V,T,R,S)$, $L(G) = \{ w \in T^* \mid S \overset{*}{\Rightarrow} w \}$

CFG (more examples)

- Let $G = (\{S\}, \{a,b\}, R, S)$, and the set of rules, R, is
  - $S \rightarrow aSb \mid SS \mid \varepsilon$

This notation is an abbreviation for

$S \rightarrow aSb$

$S \rightarrow SS$

$S \rightarrow \varepsilon$

- What will this grammar generate?
- If we think of a as "(" and b as ")", G generates all strings of properly nested parentheses

Quick Quiz

- Is the following a CFG?

$$G = \{ \{A,B\}, \{0,1\}, R, A \}$$

$$A \to 0B1 \mid A \mid 0$$

$$B \to 1B0 \mid 1$$

$$0B \to A$$

Designing CFG

- Can we design CFG for $\{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$?

- Do we know CFG for $\{0^n1^n \mid n \geq 0\}$?
- Do we know CFG for $\{1^n0^n \mid n \geq 0\}$?

Designing CFG

- CFG for the language $L1 = \{0^n1^n \mid n \geq 0\}$

  $S \rightarrow 0S1 \mid \varepsilon$

- CFG for the language $L2 = \{1^n0^n \mid n \geq 0\}$

  $S \rightarrow 1S0 \mid \varepsilon$

- CFG for $L1 \cup L2$

  $S \rightarrow S_1 \mid S_2$

  $S_1 \rightarrow 0S_11 \mid \varepsilon$

  $S_2 \rightarrow 1S_20 \mid \varepsilon$

Designing CFG

- Can we design CFG for $\{0^{2n}1^{3n} \mid n \geq 0\}$?

- Yes, by "linking" the occurrence of 0's with the occurrence of 1's
- The desired CFG is:

$$S \rightarrow 00S111 \mid \varepsilon$$

Quick Quiz

- Can we construct the CFG for the language { w | w is a palindrome } ?

Assume that the alphabet of w is {0,1}

- Examples for palindrome: 010, 0110, 001100, 01010, 1101011, …

Regular Language & CFG

Theorem: Any regular language can be described by a CFG.

How to prove? (By construction)

Regular Language & CFG

Proof: Let D be the DFA recognizing the language. Create a distinct variable V_i for each state q_i in D.

- Make V_0 the start variable of CFG

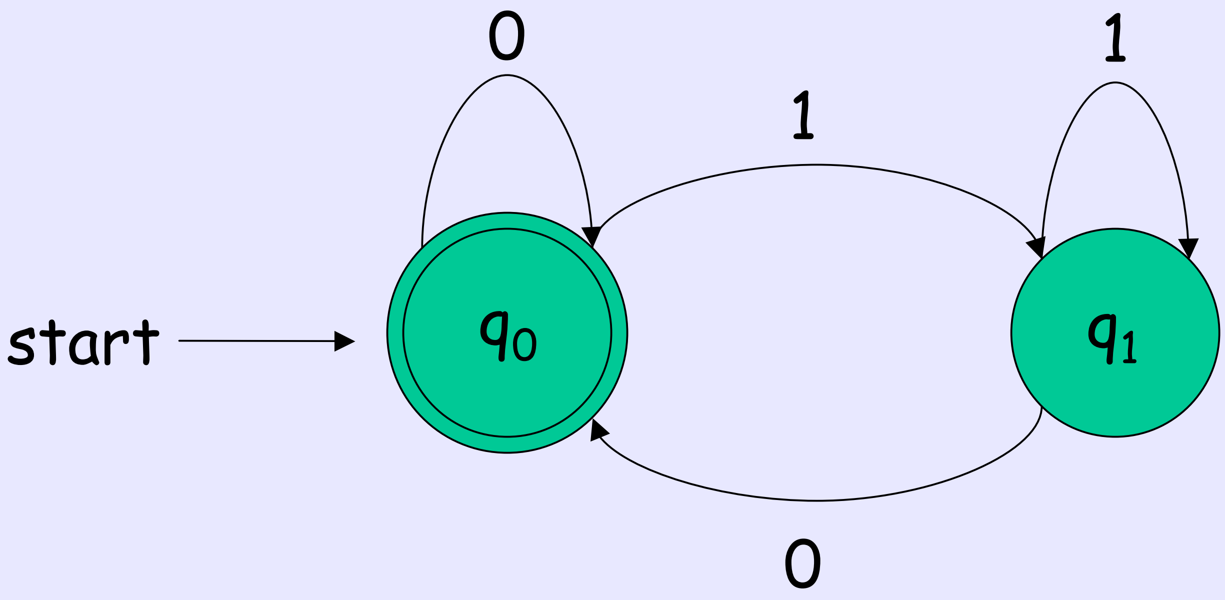
Assume that q_0 is the start state of D

- Add a rule $V_i \rightarrow aV_j$ if $\delta(q_i,a) = q_j$
- Add a rule $V_i \rightarrow \varepsilon$ if q_i is an accept state

Then, we can show that the above CFG generates exactly the same language as D (how to show?)

Regular Language & CFG
(Example)

DFA



CFG

$G = (\{V_0, V_1\}, \{0,1\}, R, V_0)$, where R is

$V_0 \rightarrow 0V_0 \mid 1V_1 \mid \varepsilon$

$V_1 \rightarrow 1V_1 \mid 0V_0$

Leftmost Derivation

- A derivation which always replace the leftmost variable in each step is called a leftmost derivation
  - E.g., Consider the CFG for the properly nested parentheses ({S}, {(,)}, R, S) with rule R: $S \rightarrow (S) \mid SS \mid \varepsilon$
  - Then, $S \Rightarrow SS \Rightarrow (S)S \Rightarrow (\,)S \Rightarrow (\,)(S)$ $\Rightarrow (\,)(\,)$ is a leftmost derivation
  - But, $S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow (\,)(S)$ $\Rightarrow (\,)(\,)$ is not a leftmost derivation
- However, we note that both derivations correspond to the same parse tree

Ambiguity

- Sometimes, a string can have two or more leftmost derivations!!
- E.g., Consider CFG ({S}, {+,x,a}, R, S) with rules R:

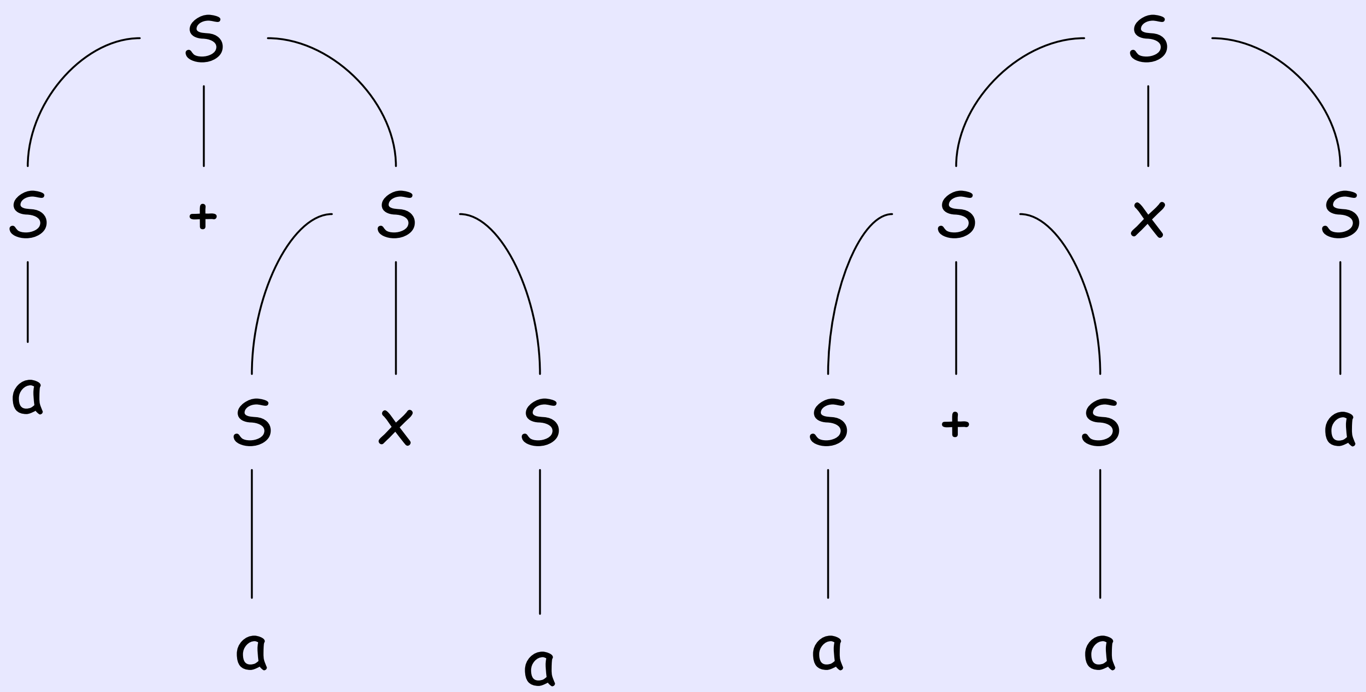
$$S \rightarrow S + S \mid S \times S \mid a$$

  - The string a + a x a has two leftmost derivations as follows:
  - $S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S \times S \Rightarrow a + a \times S$
    $\Rightarrow a + a \times a$
  - $S \Rightarrow S \times S \Rightarrow S + S \times S \Rightarrow a + S \times S \Rightarrow a + a \times S$
    $\Rightarrow a + a \times a$

Ambiguity

- If a string has two or more leftmost derivations in a CFG G, we say the string is derived ambiguously in G
- A grammar is ambiguous if some strings is derived ambiguously
- Note that the two leftmost derivations in the previous example correspond to different parse trees (see next slide)
  - In fact, each leftmost derivation corresponds to a unique parse tree

Two parse trees for a + a x a



Fun Fact: Inherently Ambiguous

- Sometimes when we have an ambiguous grammar, we can find an unambiguous grammar that generates the same language
- However, some language can only be generated by ambiguous grammar

  E.g., $\{ a^n b^n c^m \mid n, m \geq 0\} \cup \{a^n b^m c^m \mid n, m \geq 0\}$

  See last year's bonus exercise in Homework 2

- Such language is called inherently ambiguous

Chomsky Normal Form (CNF)

- A CFG is in Chomsky Normal Form if each rule is of the form

  $A \rightarrow BC$

  $A \rightarrow a$

  where
  - a is any terminal
  - A,B,C are variables
  - B, C cannot be start variable
- However, $S \rightarrow \varepsilon$ is allowed

Converting a CFG to CNF

Theorem: Any context-free language can be generated by a context-free grammar in Chomsky Normal Form.

Hint: When is a general CFG not in Chomsky Normal Form?

Proof Idea

The only reasons for a CFG not in CNF:

1. Start variable appears on right side
2. It has ε rules, such as $A \to \varepsilon$
3. It has unit rules, such as $A \to A$, or $B \to C$
4. Some rules does not have exactly two variables or one terminal on right side

Prove idea: Convert a grammar into CNF by handling the above cases

The Conversion (step 1)

- Proof: Let G be the context-free grammar generating the context-free language. We want to convert G into CNF.
- Step 1: Add a new start variable S_0 and the rule $S_0 \rightarrow S$, where S is the start variable of G

This ensures that start variable of the new grammar does not appear on right side

The Conversion (step 2)

- Step 2: We take care of all ε rules. To remove the rule $A \rightarrow \varepsilon$, for each occurrence of A on the right side of a rule, we add a new rule with that occurrence deleted.
  - E.g., $R \rightarrow uAvAw$ causes us to add the rules: $R \rightarrow uAvw$, $R \rightarrow uvAw$, $R \rightarrow uvw$
- If we have the rule $R \rightarrow A$, we add $R \rightarrow \varepsilon$ unless we had previously removed $R \rightarrow \varepsilon$

After removing $A \rightarrow \varepsilon$, the new grammar still generates the same language as G.

The Conversion (step 3)

- Step 3: We remove the unit rule $A \rightarrow B$. To do so, for each rule $B \rightarrow u$ (where u is a string of variables and terminals), we add the rule $A \rightarrow u$.
  - E.g., if we have $A \rightarrow B$, $B \rightarrow aC$, $B \rightarrow CC$, we add: $A \rightarrow aC$, $A \rightarrow CC$

After removing $A \rightarrow B$, the new grammar still generates the same language as G.

The Conversion (step 4)

- Step 4: Suppose we have a rule $A \rightarrow u_1 u_2 \ldots u_k$, where $k > 2$ and each u_i is a variable or a terminal. We replace this rule by
  - $A \rightarrow u_1 A_1$, $A_1 \rightarrow u_2 A_2$, $A_2 \rightarrow u_3 A_3$, …, $A_{k-2} \rightarrow u_{k-1} u_k$

After the change, the string on the right side of any rule is either of length 1 (a terminal) or length 2 (two variables, or 1 variable + 1 terminal, or two terminals)

The Conversion (step 4 cont.)

- To remove a rule $A \rightarrow u_1u_2$ with some terminals on the right side, we replace the terminal u_i by a new variable U_i and add the rule $U_i \rightarrow u_i$

After the change, the string on the right side of any rule is exactly a terminal or two variables

The Conversion (example)

- Let G be the grammar on the left side. We get the new grammar on the right side after the first step.

$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

The Conversion (example)

- After that, we remove $B \rightarrow \varepsilon$

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

Before removing $B \rightarrow \varepsilon$

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid a$
$A \rightarrow B \mid S \mid \varepsilon$
$B \rightarrow b$

After removing $B \rightarrow \varepsilon$

The Conversion (example)

- After that, we remove $A \rightarrow \varepsilon$

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB \mid a$

$A \rightarrow B \mid S \mid \varepsilon$

$B \rightarrow b$

Before removing $A \rightarrow \varepsilon$

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB \mid a \mid$

$SA \mid AS \mid S$

$A \rightarrow B \mid S$

$B \rightarrow b$

After removing $A \rightarrow \varepsilon$

The Conversion (example)

- Then, we remove $S \rightarrow S$ and $S_0 \rightarrow S$

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow B \mid S$

$B \rightarrow b$

After removing $S \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow B \mid S$

$B \rightarrow b$

After removing $S_0 \rightarrow S$

The Conversion (example)

- Then, we remove $A \rightarrow B$

$S_0 \rightarrow ASA \mid aB \mid a \mid$
$\quad SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid$
$\quad SA \mid AS$
$A \rightarrow B \mid S$
$B \rightarrow b$

Before removing
$A \rightarrow B$

$S_0 \rightarrow ASA \mid aB \mid a \mid$
$\quad SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid$
$\quad SA \mid AS$
$A \rightarrow b \mid S$
$B \rightarrow b$

After removing
$A \rightarrow B$

The Conversion (example)

- Then, we remove $A \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow b \mid S$
$B \rightarrow b$

Before removing $A \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$
$B \rightarrow b$

After removing $A \rightarrow S$

The Conversion (example)

- Then, we apply Step 4

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$

$B \rightarrow b$

Before Step 4

$S_0 \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$

$S \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$

$A \rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS$

$B \rightarrow b$

$A_1 \rightarrow SA$

$U \rightarrow a$

After Step 4
Grammar is in CNF

Next Time

- Pushdown Automaton (PDA)
  - An NFA equipped with a stack
- Power of CFG = Power of PDA
  - In terms of describing a language