CMSC 141 – Automata and Language Theory
Handout: Context-Free Grammars
By: Gianina Renee V. Vergara

Context Free Grammars

- A context-free grammar, CFG is a 4-tuple (V, T, P, S)...
- Each production is of the form $A \rightarrow \alpha$, where $A \in V$, $\alpha \in (V \cup T)^*$

Application

- Used in actual problems in programming languages and other areas of computer science.
- Used in the study of human language.
- Most compilers and interpreters contain a component called a parser that extracts the meaning of a program prior to generating a compiled code or performing the interpreted execution.

Example 1: CFG that tries to capture if-then-else statements. In this notation, the variables/non-terminals are those enclosed by "<>" symbols, and the terminals are in bold.

<start> → **if (**<condition>**) then {** <start> **}**
<start> → **if (**<condition>**) then {**<start>**} else {**<start>**}**
<start> → <stmt>
<condition> → **true** | **false**
<stmt> → **printf("Hello World!");**

Def'n: A **derivation** or **parse tree** is a graphic method of showing derivations of a string from the start symbol.

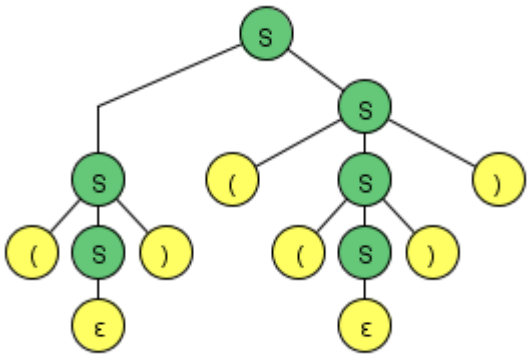
Def'n: a CFG is **ambiguous** iff it generates some sentence by two or more distinct leftmost (rightmost) derivations. Or, a CFG is ambiguous iff it generates some sentence by two or more distinct parse trees.

Example 2: Consider the grammar G = ({S}, {a,b}, P, S). The set of production is:

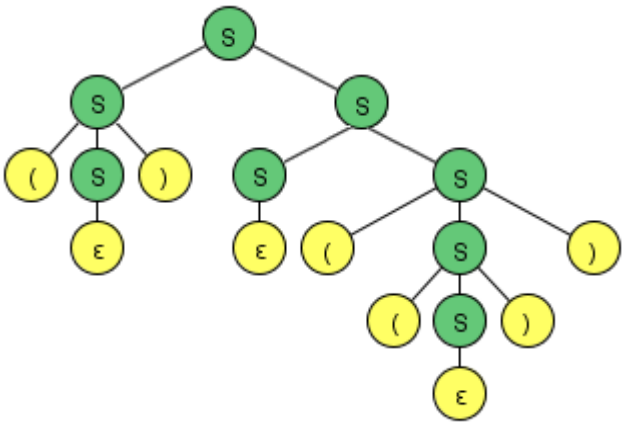
S → SS | (S) | ε

the string w = ()(()) can be produced by G using two distinct parse trees:

a.



b.



Hence, this CFG is ambiguous. L(G) is the language of all strings of properly nested parenthesis.

Def'n: Sometimes when we have an ambiguous grammar, we can find an unambiguous grammar that generates the same language. Some CFL, however, can only be generated by ambiguous grammars. Such grammars are called **inherently ambiguous**.

Brute Force Parser – Regular or Context Free Grammar with JFLAP

Multi-Input Run Grammar with JFLAP

Designing CFG's

Some basic techiniques:

- CFG's for regular languages – RL are also CFL. It will be easier to generate the CFG from the equivalent DFA of the regular language.

- Matching – enforcing the fact that your grammar generates matching pairs of symbols.
  Example 3: $L = \{0^n1^{2n} \mid n \geq 0\}$. The context-free grammar for L is
  $\mathbf{S \rightarrow 0S11 \mid \varepsilon}$
  The grammar forces every 0 to match 11.

- Using recursive relationships – when you can represent strings in the language in terms of shorter strings in the language. The variable symbol should be placed in the location where it may recursively appear.
  Example 4: L – the language of all strings of balanced brackets. Every string w in L can be viewed as either
  - uv, for $u \in L$, $v \in L$, or
  - (u), for $u \in L$.

  This gives the following grammar:
  $\mathbf{S \rightarrow SS \mid (S) \mid \varepsilon}$

- Thinking of each non-terminal A as representing a language of all strings w derivable from A. Then, combining these languages into one, using the rules.
  Example 5: $L = \{0^n1^m \mid n \neq m\}$.
  Let $L_1 = \{ 0^n1^m \mid n > m\}$, and $L_2 = \{ 0^n1^m \mid n < m\}$. Then, if S_1 generates L_1, and S_2 generates L_2, our grammar will be
  $S \rightarrow S_1 \mid S_2$
  L_1 is just the language of strings 0^n1^m with one or more extra 0's in front. So,
  $S_1 \rightarrow 0S_1 \mid 0A$
  $A \rightarrow 0A1 \mid \varepsilon$
  L_2 is just the language of strings 0^n1^m with one or more extra 1's in the end. So,
  $S_2 \rightarrow S_21 \mid A1$
  $A \rightarrow 0A1 \mid \varepsilon$
  $L = L_1 \cup L_2$
  $\mathbf{S \rightarrow S_1 \mid S_2}$
  $\mathbf{S_1 \rightarrow 0S_1 \mid 0A}$
  $\mathbf{S_2 \rightarrow S_21 \mid A1}$
  $\mathbf{A \rightarrow 0A1 \mid \varepsilon}$

Chomsky Normal Form (CNF)

- A CFG is in CNF if every rule is of the form: $A \rightarrow BC$ or $A \rightarrow a$, where a is any terminal and A, B, C are any non-terminal - *(in some books B and C may not be the start variable)*. In addition we permit the rule $S \rightarrow \varepsilon$.
  1. Ensure S does not appear on RHS, add new S_0 if necessary and copy all rules of S.
  2. Eliminate all ε rules, except $S \rightarrow \varepsilon$.
  3. Eliminate unit rules
  4. Convert rules to proper form.

- **Eliminating ε rules**:
  - Pick a ε-rule, Y → ε, and remove it.
  - Given a rule X → α where α contains n occurrences of Y , replace it with 2^n rules in which 0, …, n occurrences are replaced by ε. (Do not add X → ε if previously removed.)
    e.g.

1. Remove Y → ε..	2. Replace X with ε occurences of Y..	3. New rules..
X → aYcbY, Y → a | ε	X → aYcbY | aεcbY | aYcbε | aεcbε, Y → a	X → aYcbY | acbY | aYcb | acb, Y → a

- **Eliminating unit rules:**
  - Pick a unit rule A → B and remove it.
  - For every rule B → *u*, add rule A → *u* unless this is a unit rule that was previously removed
    e.g.

1. Remove S → A..	2. add occurrences of A to S
S → A, A → aB | b, B → aS	S → aB | b, A → aB | b, B → aS

CNF Example

	remove X→ε	remove S→X	follow format ✔		CNF
S → XX	S → XX	S → XX	S → XX	✔	S → XX
X → aXb	**S → X**	**S → aXb**	**S → AXB**		S → AB
X → ε	**S → ε**	**S → ab**	**S → AB**	✔	A → a
	X → aXb	S → ε	**A → a**	✔	B → b
	X → ab	X → aXb	**B → b**	✔	S →ε
		X → ab	S →ε	✔	X → AB
			X → AXB		**S → AZ**
			X → AB	✔	**Z → XB**
					X → AZ

[Transform Grammar with JFLAP]