CFG Ambiguity

Example:

\[
\begin{align*}
\text{Integer} & \rightarrow \text{Digit Integer} \mid \text{Digit} \\
\text{Digit} & \rightarrow 0 \mid 1 \mid 2 \mid \cdots \mid 9 \\
\text{Expression} & \rightarrow \text{Exp} + \text{Exp} \mid \text{Integer} \mid \text{Exp} \times \text{Exp}
\end{align*}
\]

Thus \(\text{Exp} \Rightarrow 3 \times 4 + 2\)

ie., \(3 \times 4 + 2 \in L(\text{Exp})\)

Observe: The same string can have multiple derivations, eg.

- **leftmost** \(\Rightarrow \text{Exp} \Rightarrow \text{Exp} + \text{Exp} \Rightarrow \text{Exp} \times \text{Exp} + \text{Exp} \Rightarrow 3 \times \text{Exp} + \text{Exp} \Rightarrow 3 \times 4 + \text{Exp} \Rightarrow 3 \times 4 + 5\), and

- **rightmost** \(\Rightarrow \text{Exp} \Rightarrow \text{Exp} + \text{Exp} \Rightarrow \text{Exp} + 2 \Rightarrow \text{Exp} \times \text{Exp} + 2 \Rightarrow \text{Exp} \times 4 + 2 \Rightarrow 3 \times 4 + 2\)

This is okay. It is usually not okay if there are multiple parse trees yielding the same string.

**Eg.**

\[
\begin{align*}
\text{Exp} & \Rightarrow \text{Exp} + \text{Exp} \mid \text{Exp} \times \text{Exp} \mid \text{Exp}
\end{align*}
\]

same yield, \(\text{Exp} \times \text{Exp} + \text{Exp}\), but very different meanings?

**Def:** A **CFG is unambiguous** if every terminal string has at most one parse tree.

Equivalently, unambiguous \(\iff\) every terminal string has at most one leftmost derivation

ambiguous \(\iff\) some terminal string has two or more parse trees (or leftmost derivations)
Removing ambiguity (leaving $L(G)$ the same)

**Idea 1**  
Operator precedence

$G$:  
$$E \rightarrow \text{Num (brackets)} \mid E + E \mid E \times E \mid (E)$$

$G'$:  
$$E \rightarrow E + E \mid E'$$  
$$E' \rightarrow \text{Num} \mid E' \times E' \mid (E)$$

"+" should have **strictly lower precedence**, i.e., be higher in the parse tree, than "*"

-but $G'$ is still ambiguous!

$E + E + E$ has parse trees

---

**Idea 2**  
Force left-associativity

$H$:  
$$E \rightarrow \text{Num} \mid E + (E)$$

-we don't want to allow this $E$

$H'$:  
$$E \rightarrow \text{Num} \mid E + \text{Num}$$

---

$G''$:  
$$E \rightarrow E + E' \mid E'$$  
$$E' \rightarrow E' \times E'' \mid E''$$  
$$E'' \rightarrow (E) \mid \text{Num}$$

-Idea: $E$ can be split into a "+"

-but $E'$ can only expand as a "*" or $(E)$

-and $E''$ can only be $(E)$ or Number.

Note: Yacc/lex can automatically deal with precedence/associativity.

---

Inherent ambiguity  
(Ch. M. U. Ch. 5.4.47)

- $L = \{a^m b^n c^n d^n \mid m, n \geq 0\} \cup \{a^m b^n c^m d^m \mid m, n \geq 0\}$

-is a context-free language (CFL)

**Theorem:** Every grammar $G$ for $L$ is ambiguous!

-essentially strings like $a^m b^n c^m d^m$ must have two parse trees, for $m$ large enough.
Simplifying the presentation of Context-free Grammars

1. Eliminate "useless" symbols
2. Convert to "Chomsky normal form" (CNF)

**Def**: A variable or terminal \( X \) is **useful** if
- \( X \) derives a terminal string, and
- \( X \) is reachable from the start, i.e.,
  \[ S \Rightarrow \alpha X \beta \]

**Eliminating useless symbols**
- Eliminate symbols that cannot generate a terminal string
- Eliminate symbols that are not reachable

**Example**

\[ G: S \rightarrow AB | C \]
- \( A \rightarrow OB | C \)
- \( B \rightarrow 1 | AO \)
- \( C \rightarrow AC | C1 \)

\[ G': S \rightarrow AB \]
- \( A \rightarrow OB, B \rightarrow 1 | AO \)

<table>
<thead>
<tr>
<th>round</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symbols: 0 1 2 3 4 5

**Algorithm**: Loop through the rules
- Every terminal is generating
- If \( A \rightarrow X_1 X_2 \cdots X_k \) and each \( X_j \) is generating, then so is \( A \)

**Updating the symbols in each round**: Until no changes are made in a loop.

**Use a graph traversal algorithm**
- \( S \) is reachable
- If \( A \) is reachable and \( A \rightarrow X_1 X_2 \cdots X_k \), then so are the \( X_j \)

**Important**: Remember to eliminate non-generating symbols first.

**Example**: \( S \rightarrow AB | a \)  
- \( A \rightarrow 1 \)  

Should simplify to \( S \rightarrow a \)

In general, if \( S \Rightarrow CD \) and \( C \) is not generating, then \( D \) is reachable, but this derivation can't go anywhere.
**Chomsky Normal Form (CNF)**

**Definition:** A context-free grammar is in Chomsky normal form if all its productions are of the form

\[ A \rightarrow BC \text{ or } A \rightarrow a \]

where \( A, B, C \) are variables and \( a \) is a terminal.

**Theorem:** For any context-free language \( L \), there exists a context-free grammar \( G \) with
- \( L(G) = L \setminus \{ \varepsilon \} \)
- \( G \)'s symbols are all useful
- \( G \) is in Chomsky Normal Form.

**Example:**

\[
E \rightarrow E + E \mid E * E \mid \text{Digit} \\
\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9
\]

in CNF:

\[
E \rightarrow E \text{ Eplus} \mid E \text{ Etimes} \mid 0 \mid 1 \mid \ldots \mid 9 \\
\text{Eplus} \rightarrow \text{PLUS} \ E \\
\text{Etimes} \rightarrow \text{TIMES} \ E \\
\text{PLUS} \rightarrow + \\
\text{TIMES} \rightarrow *
\]

**Why?**
- Simpler because every production has one of two forms
- For proving the Pumping Lemma for context-free languages

**Proof Idea:**
1. Eliminate useless symbols
2. Eliminate \( E \) productions \((L \rightarrow L \setminus \{ \varepsilon \})\)
   - Find "nullable" symbols \( A \Rightarrow E \), and replace them
   - If \( A \rightarrow BC \) are \( B, C \) are nullable,
     - then add \( A \rightarrow BC \mid B \mid C \)
     - throw away \( A \rightarrow E \) productions
3. Eliminate unit productions
4. Convert to Chomsky Normal Form by splitting larger productions, adding new variables