Context Free Grammars

Shakir Ullah Shah

Example

$S \rightarrow aSa \mid aBa$

$B \rightarrow bB \mid b$

First production builds equal number of a's on both sides and recursion is terminated by $S \rightarrow aBa$

Recursion of $B \rightarrow bB$ may add any number of b's and terminates with $B \rightarrow b$

$L(G) = \{a^n b^m a^n \; n>0, m>0\}$

Example

Consider the following CFG $\Sigma = \{a, b\}$

$S \rightarrow aXb \mid bXa \mid \Lambda$

$X \rightarrow aX \mid bX \mid \Lambda$

The above CFG generates the language of strings, defined over $\Sigma = \{a,b\}$, beginning and ending in different letters OR including Λ as well.

Examples

A grammar that generates the language consisting of even-length string over {a, b}

$S \to aO \mid bO \mid \Lambda$

$O \to aS \mid bS$

G1: $S \to AB$

$A \to aA \mid a$

$B \to bB \mid \Lambda$

And

G2: $S \to aS \mid aB$

$B \to bB \mid \Lambda$

Both generates a^+b^*

Even-Even

Devise a grammar that generates strings with even number of a’s and even number of b’s

We can use the strategy of previous example i.e. non-terminal represents the current states of the derived string

The non-terminals with interpretations are:

Non-Terminal	Interpetation
S	Even a’s and Even b’s
A	Even a’s and Odd b’s
B	Odd a’s and Even b’s
C	Odd a’s and Odd b’s

Even-Even contd.

The grammar will be

$S \rightarrow aB \mid bA \mid \Lambda$

$A \rightarrow aC \mid bS$

$B \rightarrow aS \mid bC$

$C \rightarrow aA \mid bB$

When S is present, the derived string satisfies the Even-Even condition as $S \rightarrow \Lambda$

Construct a CFG over {a, b} generating all strings that do not contain *abc?*

Remarks

We have seen that some regular languages can be generated by CFGs, and some non-regular languages can also be generated by CFGs.

ALL regular languages can be generated by CFGs.

There is some non-regular language that cannot be generated by any CFG.

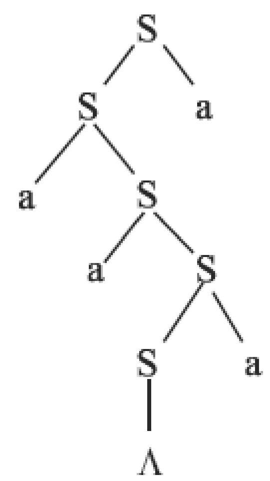
Thus, the set of languages generated by CFGs is properly **larger** than the set of regular languages, but properly **smaller** than the set of all possible languages.

Parse Tree

We can use a tree diagram to show that derivation process:
We start with the starting symbol S**. Every time we use a production to replace a nonterminal by a string, we draw downward lines from the nonterminal to EACH character in the string.**

1. $S \rightarrow Sa$
2. $S \rightarrow aS$
3. $S \rightarrow \Lambda$

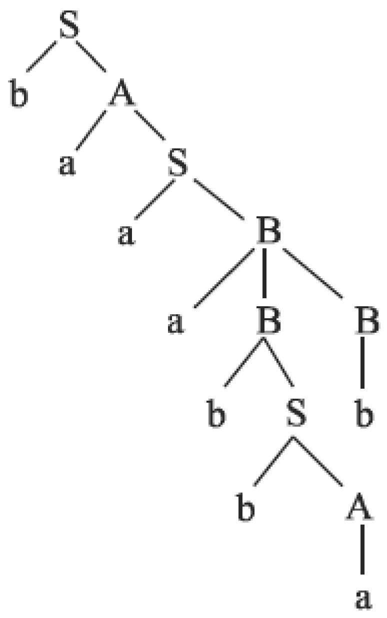
$$
\begin{aligned}
S &\Rightarrow Sa && \text{(Rule 1)} \\
&\Rightarrow aSa && \text{(Rule 2)} \\
&\Rightarrow aaSa && \text{(Rule 2)} \\
&\Rightarrow aaSaa && \text{(Rule 1)} \\
&\Rightarrow aa\Lambda aa && \text{(Rule 3)} \\
&= aaaa
\end{aligned}
$$



Tree diagrams are also called **syntax trees**, **parse trees**, **generation trees**, **production trees**, or **derivation trees.**

1. $S \rightarrow \Lambda$
2. $S \rightarrow bA$
3. $S \rightarrow aB$
4. $A \rightarrow a$
5. $A \rightarrow aS$
6. $A \rightarrow bAA$
7. $B \rightarrow b$
8. $B \rightarrow bS$
9. $B \rightarrow aBB$

$S \Rightarrow bA$	(Rule 2)
$\Rightarrow baS$	(Rule 5)
$\Rightarrow baaB$	(Rule 3)
$\Rightarrow baaaBB$	(Rule 9)
$\Rightarrow baaaBb$	(Rule 7)
$\Rightarrow baaabSb$	(Rule 8)
$\Rightarrow baaabbAb$	(Rule 2)
$\Rightarrow baaabbab$	(Rule 4)



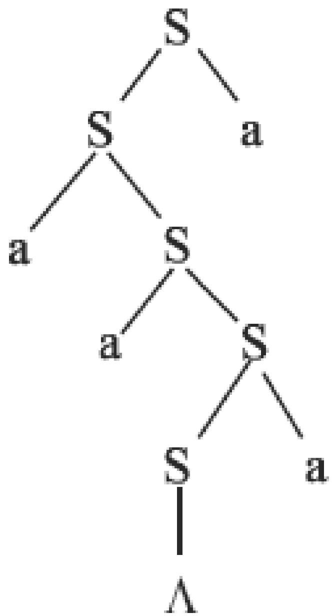
Ambiguity

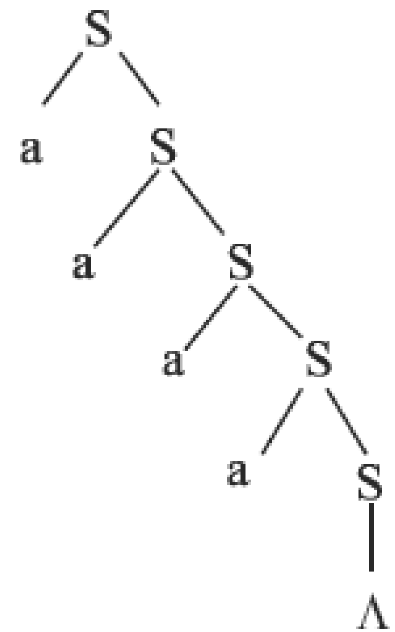
A CFG is called ambiguous if there is one word it generates which has two different parse trees.

If a CFG is not ambiguous, it is called unambiguous.

Ambiguity

$$S \rightarrow aS \mid Sa \mid \Lambda$$



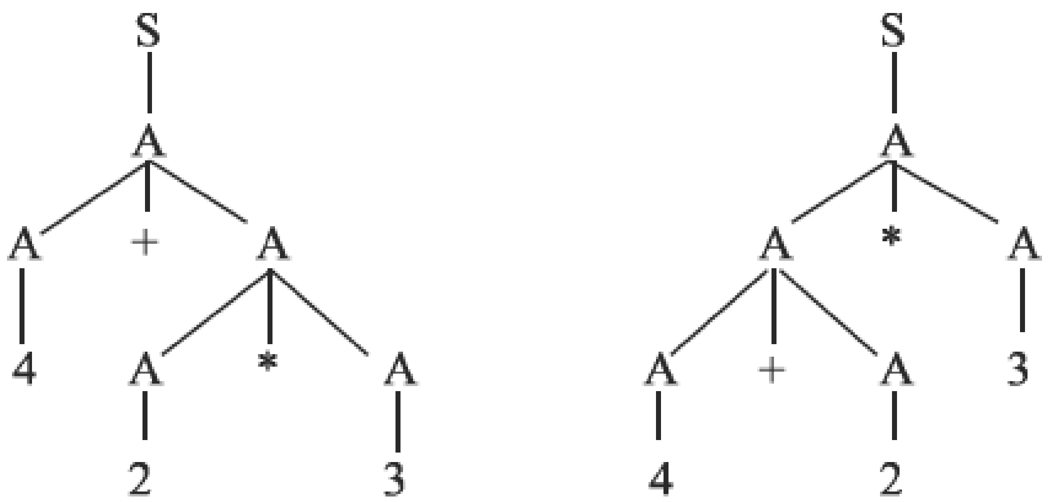


1. $S \rightarrow aS$
2. $S \rightarrow \Lambda$

Arithmetic Expressions

$S \rightarrow A$

$A \rightarrow \text{integer} \mid A + A \mid A - A \mid A * A \mid A / A \mid (A)$



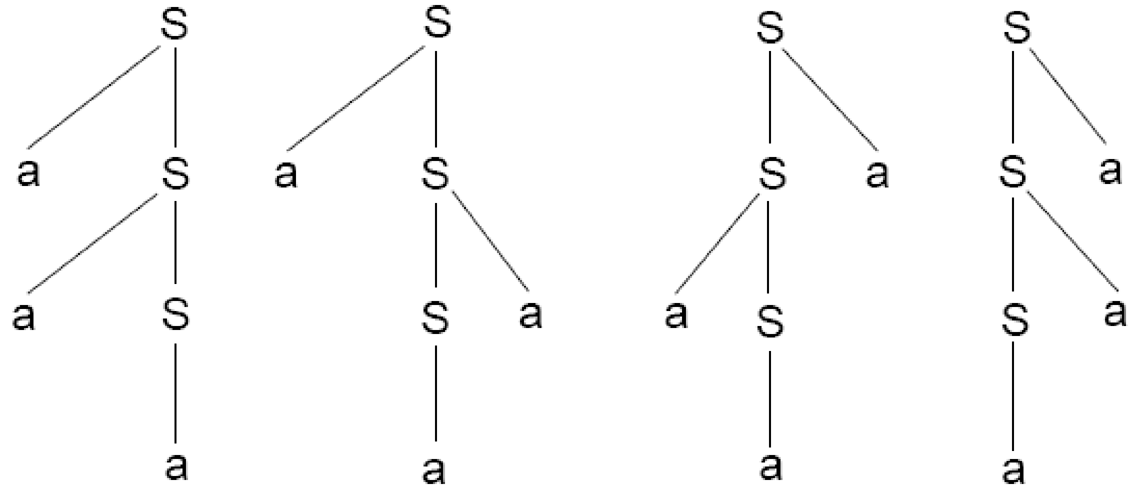
4 + 2*3

Ambiguity- example

The following CFG defines the language of all non-null strings of a's:

$$S \rightarrow aS \mid Sa \mid a$$

The word a^3 can be generated by 4 differer



Ambiguity- example

Consider the language generated by the following CFG:

$S \rightarrow AB$

$A \rightarrow a$

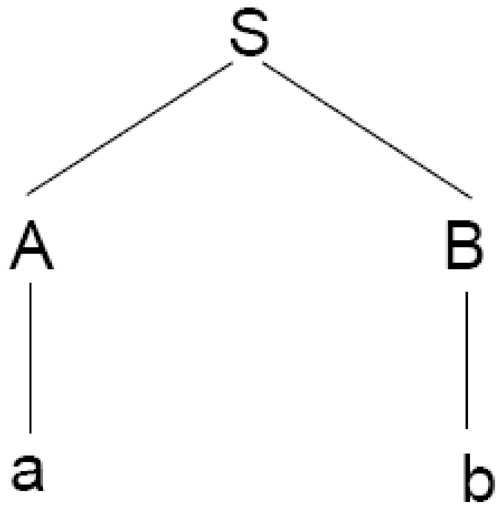
$B \rightarrow b$

There are two derivations of the word ab:

S => AB => aB => ab

or S => AB => Ab => ab

However, These two derivations correspond to the same syntax tree:



The word ab is therefore not ambiguous. In general, when all the possible derivation trees are the same for a given word, then the word is unambiguous.

Leftmost derivation

If at each step in a derivation a production is applied to leftmost variable, then derivation is said to be leftmost

Rightmost derivation

If at each step in a derivation a production is applied to rightmost variable, then derivation is said to be rightmost

*4 + 2*3*

$$
\begin{aligned}
S &\rightarrow E \\
E &\rightarrow T + E \mid T - E \mid T \\
T &\rightarrow F * T \mid F / T \mid F \\
F &\rightarrow \text{integer} \mid (E)
\end{aligned}
$$

Leftmost Derivation

$$
\begin{aligned}
S &\Rightarrow E \\
&\Rightarrow \mathbf{T} + E \\
&\Rightarrow \mathbf{F} + E \\
&\Rightarrow 4 + E \\
&\Rightarrow 4 + \mathbf{F}*T \\
&\Rightarrow 4 + 2*T \\
&\Rightarrow 4 + 2*F \\
&\Rightarrow 4 + 2*3
\end{aligned}
$$

$$4 + 2*3$$

$$
\begin{aligned}
S &\rightarrow E \\
E &\rightarrow T + E \mid T - E \mid T \\
T &\rightarrow F * T \mid F / T \mid F \\
F &\rightarrow \text{integer} \mid (E)
\end{aligned}
$$

Rightmost Derivation

$$
\begin{aligned}
S &\Rightarrow E \\
&\Rightarrow T + \mathbf{E} \\
&\Rightarrow T + F*\mathbf{T} \\
&\Rightarrow T + F*\mathbf{F} \\
&\Rightarrow T + \mathbf{F}*3 \\
&\Rightarrow \mathbf{T} + 2*3 \\
&\Rightarrow F + 2*3 \\
&\Rightarrow 4 + 2*3
\end{aligned}
$$