CS 302.1 - Automata Theory

Lecture 05

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Quick Recap

Grammars: A Grammar is a 4-tuple (V, Σ, P, S), such that

- V is the set of **Variables**
- Σ is the set of **Terminals**
- P is the set of production **Rules** [$(\boldsymbol{V} \cup \boldsymbol{T})^{*}\boldsymbol{V}(\boldsymbol{V} \cup \boldsymbol{T})^{*} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^{*}$]
- S is the **Start Variable** [The variable in the LHS of the first rule is generally the start variable]

- To show that a string $\boldsymbol{w} \in \boldsymbol{L}(\boldsymbol{G})$, we show that there exists a **derivation ending up in $\boldsymbol{w}$** $(\boldsymbol{S} \overset{*}{\Rightarrow} \boldsymbol{w})$.
- The **language of the grammar, $\boldsymbol{L}(\boldsymbol{G})$** is $\{w \in \Sigma^{*} | S \overset{*}{\Rightarrow} w\}$

Right Linear grammar: If the *rules* of the underlying grammar G are of the form

$$Var \rightarrow Ter\ \boldsymbol{Var}$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

then it is **Right-linear grammar.**

Left linear grammar: If the *rules* of the underlying grammar G are of the form

$$Var \rightarrow\ \boldsymbol{Var}\ Ter$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

then such a grammar is called **Left-linear grammar.**

Left-linear grammar $\equiv$ Right-linear grammar $\equiv$ DFA $\equiv$ NFA $\equiv$ Regular Expressions

Quick Recap

Grammars: A Grammar is a 4-tuple (V, Σ, P, S), such that

- V is the set of **Variables**
- Σ is the set of **Terminals**
- P is the set of production **Rules** [$(\boldsymbol{V} \cup \boldsymbol{T})^* \boldsymbol{V} (\boldsymbol{V} \cup \boldsymbol{T})^* \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$]
- S is the **Start Variable** [The variable in the LHS of the first rule is generally the start variable]

- To show that a string $\boldsymbol{w} \in \boldsymbol{L}(\boldsymbol{G})$, we show that there exists a **derivation ending up in $\boldsymbol{w}$** $(\boldsymbol{S} \overset{*}{\Rightarrow} \boldsymbol{w})$.
- The **language of the grammar, $\boldsymbol{L}(\boldsymbol{G})$** is $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$

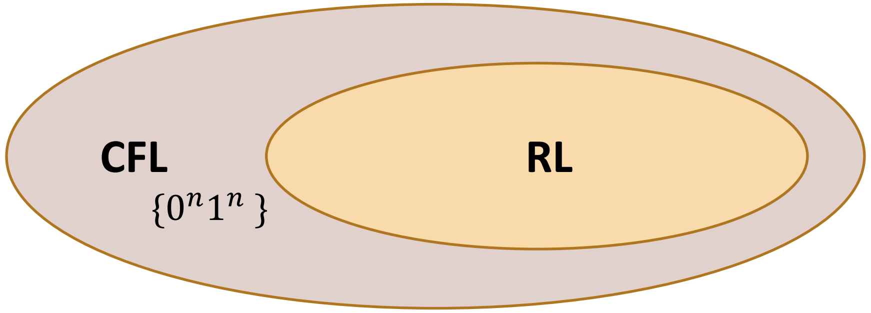
Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**

$$L(G) = \{\omega | \omega = 0^n 1^n, n \geq 0\}$$

So although $\boldsymbol{L}(\boldsymbol{G})$ is not regular, it is context-free.



Context free Grammars

Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \to (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**

Regular languages $\subset$ **Context Free Languages**.

Consider the Grammar G with the following rules:

$$S \to 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$S \to \epsilon$$

$$\{\epsilon\}$$

Context free Grammars

Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**

Regular languages ⊂ **Context Free Languages**.

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$S \rightarrow 0\boldsymbol{S}1 \rightarrow 0\boldsymbol{0S1}1 \ldots$$

$$\{\epsilon, 01, 0011, \ldots 0^n1^n\}$$

Context free Grammars

Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**

Regular languages $\subset$ **Context Free Languages**.

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$S \rightarrow \boldsymbol{0S1} \rightarrow 0\boldsymbol{SS}1 \rightarrow 0\boldsymbol{0S1}S1 \rightarrow 001S1 \rightarrow 001\boldsymbol{0S1}1 \rightarrow 001011$$

$$\{\epsilon, 01, 0011, \ldots 0^n1^n, 001011, \ldots.\}$$

Context free Grammars

Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**

Regular languages $\subset$ **Context Free Languages**.

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$S \rightarrow \boldsymbol{0S1} \rightarrow 0\boldsymbol{SS}1 \rightarrow 0\boldsymbol{0S1}S1 \rightarrow 001S1 \rightarrow 001\boldsymbol{0S1}1 \rightarrow 001011$$

$$\{\epsilon, 01, 0011, \ldots 0^n 1^n, 001011, \ldots.\}$$

Show that the string $\mathbf{010101} \in \boldsymbol{L}(\boldsymbol{G})$.

Context free Grammars

Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**

Regular languages $\subset$ **Context Free Languages**.

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{SS}S \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}SS \rightarrow 0S1\mathbf{0}\boldsymbol{S}\mathbf{1}S \rightarrow 0S10S1\mathbf{0}\boldsymbol{S}\mathbf{1} \rightarrow 010101$$

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots .\}$$

Context free Grammars

Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**

Regular languages $\subset$ **Context Free Languages**.

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{SSS} \rightarrow \boldsymbol{0S1}SS \rightarrow 0S1\boldsymbol{0S1}S \rightarrow 0S10S1\boldsymbol{0S1} \rightarrow 010101$$

$$\{\epsilon, 01, 0011, \ldots 0^n1^n, 001011, 010101, \ldots.\}$$

What is $\boldsymbol{L(G)}$?

Context free Grammars

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \ldots 0^n 1^n, 001011, 010101, \ldots.\}$$

What is $L(G)$?

Context free Grammars

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \ldots 0^n1^n, 001011, 010101, \ldots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Context free Grammars

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots.\}$

$$\{\epsilon, ()\}$$

Context free Grammars

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots .\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots .\}$

$$\{\epsilon, (\quad), ((\quad)), \dots ., \}$$

Context free Grammars

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots.\}$

$$\left\{\epsilon, (\quad), ((\quad)), \dots, \left(\left(((\dots.))\right)\right)\right\}$$

Context free Grammars

Consider the Grammar G with the following rules:

$$S \to 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots.\}$

$$\left\{\epsilon, (\quad), ((\quad)), \dots, \left(\left(((\dots.))\right)\right), ((\quad)(\quad)), \dots.\right\}$$

Context free Grammars

Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \ldots 0^n1^n, 001011, 010101, \ldots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \ldots, 0^n1^n, 001011, 010101, \ldots.\}$

$$\left\{\epsilon, (\ \ ), ((\ \ )), \ldots, \left(\left(\left((\ldots.)\right)\right)\right), ((\ \ )(\ \ )), ()()(), \ldots.\right\}$$

So, $L(G)$ is the language of all strings of properly nested parentheses.

$$L(G) = \{\omega | \omega \text{ is a correctly nested parenthesis}\}$$

Context free Grammars

Constructing CFG corresponding to a Language.

There is no fixed recipe for doing this. Requires some level of creativity.

Some tips might come in handy:

- Check if the CFL is a union of simpler languages. If $L(G) = L(G_1) \cup L(G_2)$ and G_1 and G_2 are known. If S_1 is the start variable for G_1 and S_2 is the start variable for G_2 then the the rules of G:

$$
\begin{aligned}
S &\rightarrow S_1 | S_2 \\
S_1 &\rightarrow \cdots\cdots \\
S_2 &\rightarrow \cdots\cdots
\end{aligned}
$$

Context free Grammars

Constructing CFG corresponding to a Language.

There is no fixed recipe for doing this. Requires some level of creativity.

Some tips might come in handy:

- Check if the CFL is a union of simpler languages. If $L(G) = L(G_1) \cup L(G_2)$ and G_1 and G_2 are known. If S_1 is the start variable for G_1 and S_2 is the start variable for G_2 then the the rules of G:

$$S \rightarrow S_1 | S_2$$
$$S_1 \rightarrow \cdots\cdots$$
$$S_2 \rightarrow \cdots\cdots$$

- Grammars with rules such as $S \rightarrow aSb$ help generate strings where the corresponding machine would need unbounded memory to *remember* the number of *a's* needed to verify that there are an equal number of *b's*. This was not possible with regular expressions/linear grammars.

Context free Grammars

Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as $S \rightarrow aSb$ help generate where the portions of *a* and *b* are equal.

Example: Construct the grammar G such that $\boldsymbol{L(G) = \{\omega | \omega}$ **has equal number of** $\boldsymbol{0}$**'s and** $\boldsymbol{1}$**'s**$\}$

Context free Grammars

Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as $S \rightarrow aSb$ help generate where the portions of *a* and *b* are equal.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega}|\boldsymbol{\omega}$ **has equal number of** $\mathbf{0}$**'s and** $\mathbf{1}$**'s**$\}$

- The first thing to notice is that $L_1 = \{0^n 1^n, n \geq 0\} \subset L(G)$. We know the grammar for this language.
- Any string $\omega \in L_1$ has a series of 0's followed by an equal number of 1's.
- Again, consider L_2 to comprise all strings that start with a series of 1's followed by an equal number of 0's, i.e.

$$L_2 = \{1^n 0^n, n \geq 0\}$$

- The grammar for L_2 is similar to that of L_1: replace the 0's with 1's and vice versa. Importantly, $L_2 = \{1^n 0^n, n \geq 0\} \subset L(G)$ also.

- Also, $L_1 \cup L_2 \subset L(G)$

Context free Grammars

Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as $S \rightarrow aSb$ help generate where the portions of *a* and *b* are equal.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega}|\boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

- So $\mathrm{L}'(\mathrm{G}') = \{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\} \subset L(G)$

- Grammar for L_1: $S \rightarrow 0S1|\epsilon$

- Grammar for L_2: $S \rightarrow 1S0|\epsilon$

- Grammar for $L_1 \cup L_2$:

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$

Context free Grammars

Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as $S \rightarrow aSb$ help generate where the portions of *a* and *b* are equal.

Example: Construct the grammar G such that $\boldsymbol{L(G) = \{\omega | \omega}$ **has equal number of** $\mathbf{0}$**'s and** $\mathbf{1}$**'s**$\}$

- Grammar for $L_1 \cup L_2$:

$$S \rightarrow S_1 | S_2$$
$$S_1 \rightarrow 0S_1 1 | \epsilon$$
$$S_2 \rightarrow 1S_2 0 | \epsilon$$

- **Is that all? Is** $\boldsymbol{L_1 \cup L_2 = L(G)}$? $L_1 \cup L_2$ contains all strings that have equal number 0's followed by equal number of 1's or vice versa.

Context free Grammars

Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as $S \rightarrow aSb$ help generate where the portions of *a* and *b* are equal.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega}|\boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

- Grammar for $L_1 \cup L_2$:

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$

- **Is that all? Is $\boldsymbol{L_1 \cup L_2 = L(G)}$?** $L_1 \cup L_2$ contains all strings that have equal number 0's followed by equal number of 1's or vice versa.
- What about strings such as $s_1 = 0101\cdots$ and $s_2 = 1010\cdots$? For this we need to be able to go from

$$0\boldsymbol{S_1}1 \rightarrow 0\boldsymbol{S_2}1 \rightarrow 0\boldsymbol{1S_20}1 \rightarrow \cdots$$

Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega} | \boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

- Grammar for $L_1 \cup L_2$:

$$S \to S_1 | S_2$$
$$S_1 \to 0S_11 | \epsilon$$
$$S_2 \to 1S_20 | \epsilon$$

- What about strings such as $s_1 = 0101 \cdots$ and $s_2 = 1010 \cdots$? Add transitions $S_1 \to S_2$ and $S_2 \to S_1$.

$$S \to S_1 | S_2$$
$$S_1 \to 0S_11 | \epsilon$$
$$S_2 \to 1S_20 | \epsilon$$
$$S_1 \to S_2$$
$$S_2 \to S_1.$$

Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega}|\boldsymbol{\omega}$ **has equal number of $\mathbf{0}$'s and $\mathbf{1}$'s**$\}$

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$
$$S_1 \rightarrow S_2$$
$$S_2 \rightarrow S_1$$

- Can't we simplify this? We can replace S_1 and S_2 with a single Start variable as follows: $\boldsymbol{S \rightarrow 0S1|1S0|\epsilon}$
- What kind of strings does the grammar generate? Well if we use Rule $\boldsymbol{S \rightarrow 0S1}$, m times, we get to rules such as 0^mS1^m.
- Now applying the rule $S \rightarrow 1S0$, k times, we get $\boldsymbol{0^m1^kS0^k1^m}$.
- So the strings we obtain are of the form:

$$\{0^{m_1}1^{n_1}0^{m_2}1^{n_2}\cdots 0^{n_2}1^{m_2}0^{n_1}1^{m_1}\}\cup\{1^{m_1}0^{n_1}1^{m_2}0^{n_2}\cdots 1^{n_2}0^{m_2}1^{n_1}0^{m_1}\} \in L(G)$$

Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega}|\boldsymbol{\omega}$ **has equal number of 0’s and 1’s**$\}$

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$
$$S_1 \rightarrow S_2$$
$$S_2 \rightarrow S_1$$

- Simplified grammar:

$$S \rightarrow 0S1|1S0|\epsilon$$

Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega} | \boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

$$S \to S_1 | S_2$$
$$S_1 \to 0S_1 1 | \epsilon$$
$$S_2 \to 1S_2 0 | \epsilon$$
$$S_1 \to S_2$$
$$S_2 \to S_1$$

- Simplified grammar:

$$S \to 0S1 | 1S0 | \epsilon$$

- Is that all? What about strings such as $\{\mathbf{0110}, \mathbf{00111100}\}$?

- More generally, what about strings that are a concatenation of L_1 and L_2?

Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega}|\boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$
$$S_1 \rightarrow S_2$$
$$S_2 \rightarrow S_1$$

- Simplified grammar:

$$S \rightarrow 0S1|1S0|\epsilon$$

- Is that all? What about strings such as $\{\mathbf{0110}, \mathbf{00111100}\}$?
- More generally, what about strings that are a concatenation of L_1 and L_2?
- Adding transitions like $S \rightarrow S_1S_2$ incorporates this.

Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar G such that $\boldsymbol{L(G)} = \{\boldsymbol{\omega} | \boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

$$S \rightarrow S_1 | S_2 | S_1 S_2$$
$$S_1 \rightarrow 0 S_1 1 | \epsilon$$
$$S_2 \rightarrow 1 S_2 0 | \epsilon$$
$$S_1 \rightarrow S_2$$
$$S_2 \rightarrow S_1$$

- Simplify this further.

G: $\boldsymbol{S \rightarrow SS | 0S1 | 1S0 | \epsilon}$

Parse trees for CFG

Consider the Grammar G with the following rules:

$$S \to 0S1|SS|\epsilon$$

One derivation:

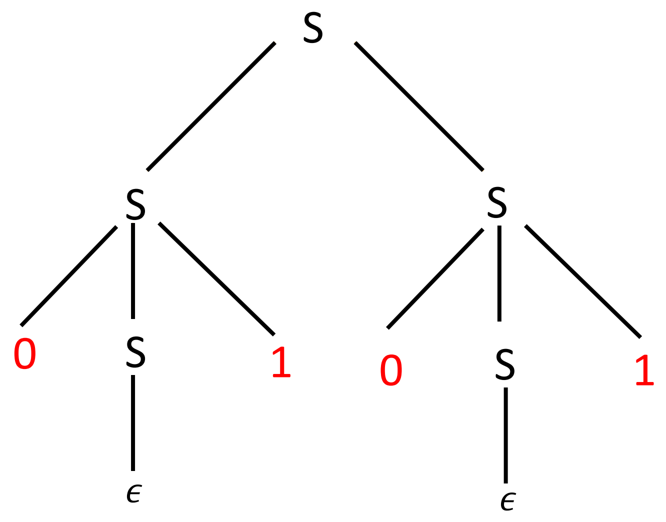
$$S \to \boldsymbol{SS} \to \mathbf{0}\boldsymbol{S}\mathbf{1}S \to 0S1\mathbf{0}\boldsymbol{S}\mathbf{1} \to \mathbf{0101}$$

Parse trees: These are ordered trees that provide alternative representations of the derivation of a grammar.

Parsing is a useful technique for compilers (Analysis of syntax eg: take sequence of tokens as input & output parse trees which provides structural representation of the input while checking for the correct syntax).

Features:

- The root node is the **Start variable**
- Branch out to nodes of the next level by following any of the rules of the grammar
- Stop when all the leaf nodes of the tree are terminals
- Read the terminals in the leaves from left to right.
- If w is the string obtained, then $\boldsymbol{S} \stackrel{*}{\Rightarrow} \boldsymbol{w}$ and $\boldsymbol{w} \in \boldsymbol{L}(\boldsymbol{G})$



Parse trees for CFG

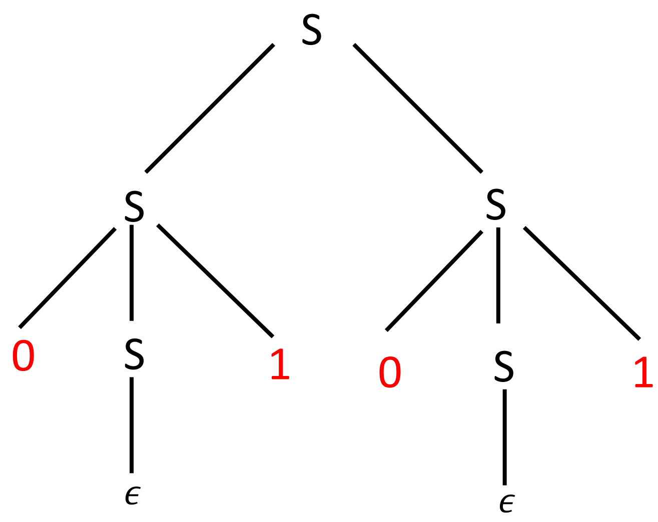
Consider the Grammar G with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Consider the following derivations for 0101:

1. $S \rightarrow \boldsymbol{SS} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}S \rightarrow 0S1\mathbf{0}\boldsymbol{S}\mathbf{1} \rightarrow \mathbf{0101}$
2. $S \rightarrow \boldsymbol{SS} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}S \rightarrow 01S \rightarrow 01\mathbf{0}\boldsymbol{S}\mathbf{1} \rightarrow \mathbf{0101}$
3. $S \rightarrow \boldsymbol{SS} \rightarrow S\mathbf{0}\boldsymbol{S}\mathbf{1} \rightarrow S01 \rightarrow \mathbf{0}\boldsymbol{S}101 \rightarrow \mathbf{0101}$

- The parse trees for all these derivations are the same.



Parse trees for CFG

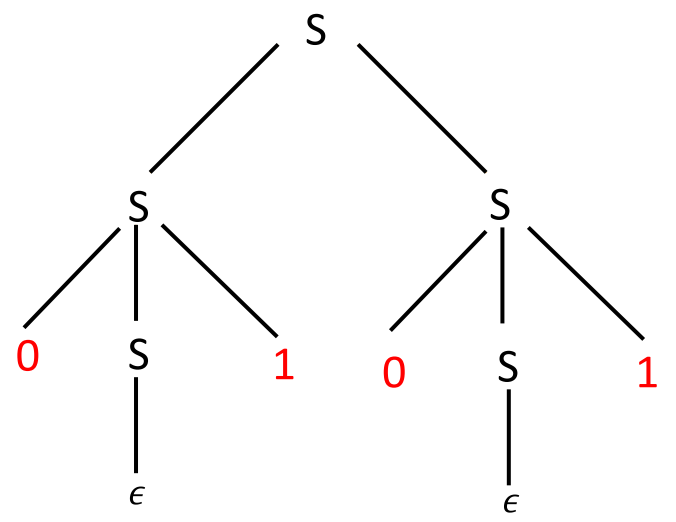
Consider the Grammar G with the following rules:

$$S \to 0S1|SS|\epsilon$$

Consider the following derivations for 0101:

1. $S \to \boldsymbol{SS} \to \boldsymbol{0S1}S \to 0S1\boldsymbol{0S1} \to \mathbf{0101}$
2. $S \to \boldsymbol{SS} \to \boldsymbol{0S1}S \to 01S \to 01\boldsymbol{0S1} \to \mathbf{0101}$
3. $S \to \boldsymbol{SS} \to S\boldsymbol{0S1} \to S01 \to \boldsymbol{0S}101 \to \mathbf{0101}$

- The parse trees for all these derivations are the same.
- If a string is derived by replacing only the leftmost variable at every step, then the derivation is a **leftmost derivation.** (e.g. derivation 2.)
- ……………….rightmost variable = **rightmost derivation** (e.g. derivation 3.)
- Derivations may not always be **leftmost** or **rightmost** (e.g. derivation 1.)



Parse trees for CFG

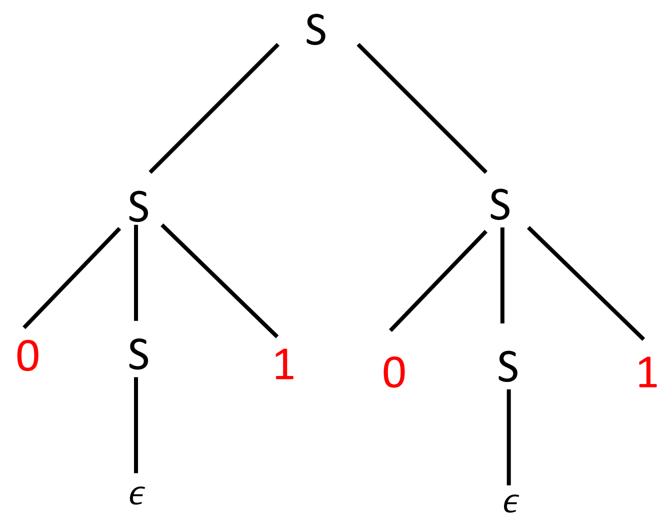
Consider the Grammar G with the following rules:

$$S \to 0S1|SS|\epsilon$$

Consider the following derivations for 0101:

1. $S \to \boldsymbol{SS} \to \mathbf{0}\boldsymbol{S}\mathbf{1}S \to 0S1\mathbf{0}\boldsymbol{S}\mathbf{1} \to \mathbf{0101}$
2. $S \to \boldsymbol{SS} \to \mathbf{0}\boldsymbol{S}\mathbf{1}S \to 01S \to 01\mathbf{0}\boldsymbol{S}\mathbf{1} \to \mathbf{0101}$
3. $S \to \boldsymbol{SS} \to S\mathbf{0}\boldsymbol{S}\mathbf{1} \to S01 \to \mathbf{0}\boldsymbol{S}101 \to \mathbf{0101}$

- The parse trees for all these derivations are the same.
- If a string is derived by replacing only the leftmost variable at every step, then the derivation is a **leftmost derivation.** (e.g. derivation 2.)
- ………………..rightmost variable = **rightmost derivation** (e.g. derivation 3.)
- Derivations may not always be **leftmost** or **rightmost** (e.g. derivation 1.)



Ambiguous grammars: A CFG G is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$, such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Parse trees for CFG

Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L(G)}$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \rightarrow 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega = 010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**

Parse trees for CFG

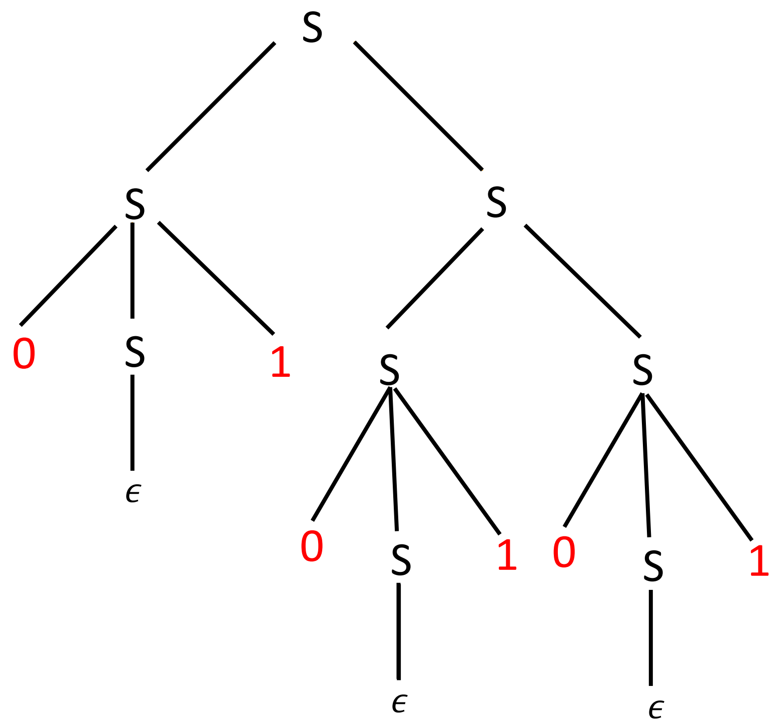
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}|\boldsymbol{SS}|\boldsymbol{\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\ \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{SS}$

Parse trees for CFG

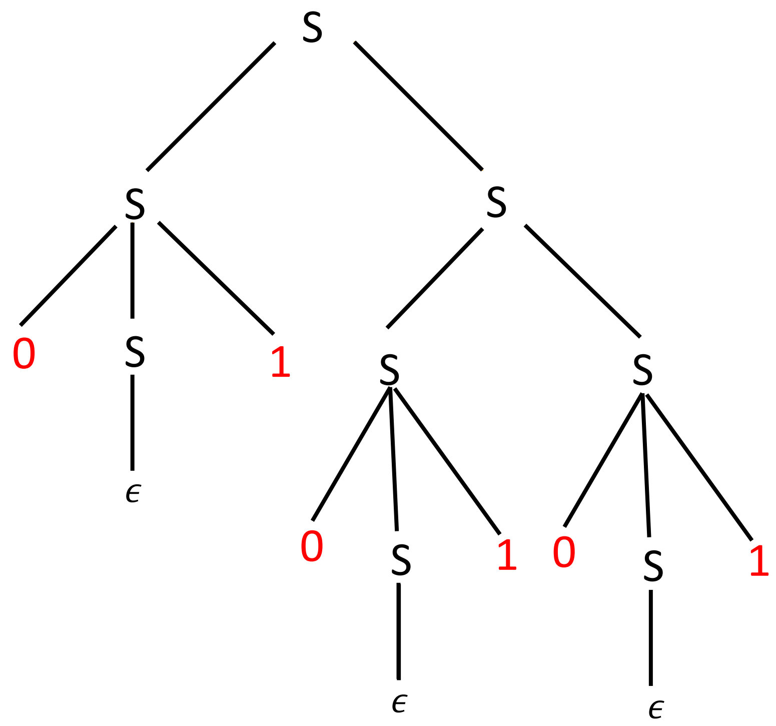
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$, such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \to 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega = 010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \to \boldsymbol{SS}$

Parse trees for CFG

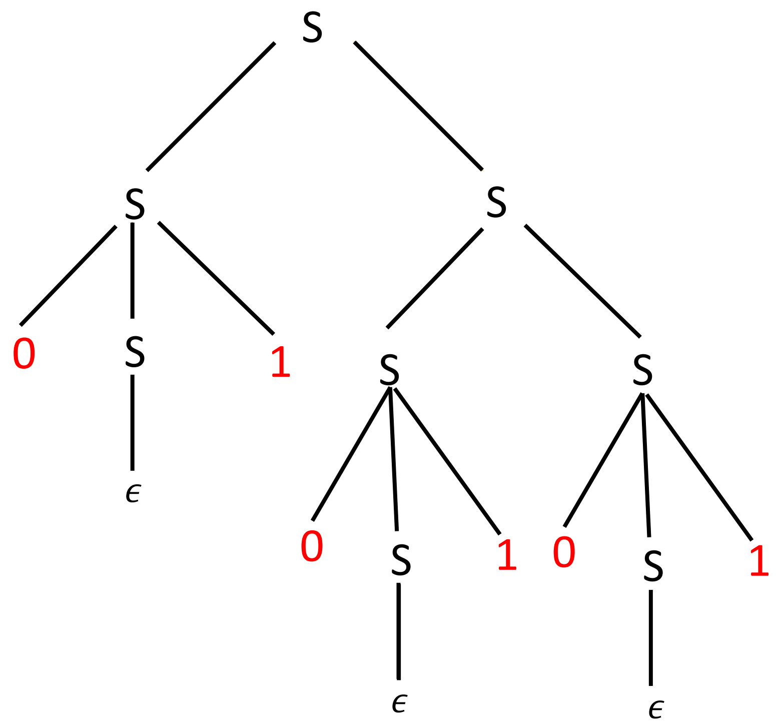
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}|\boldsymbol{SS}|\boldsymbol{\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{SS} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}S$

Parse trees for CFG

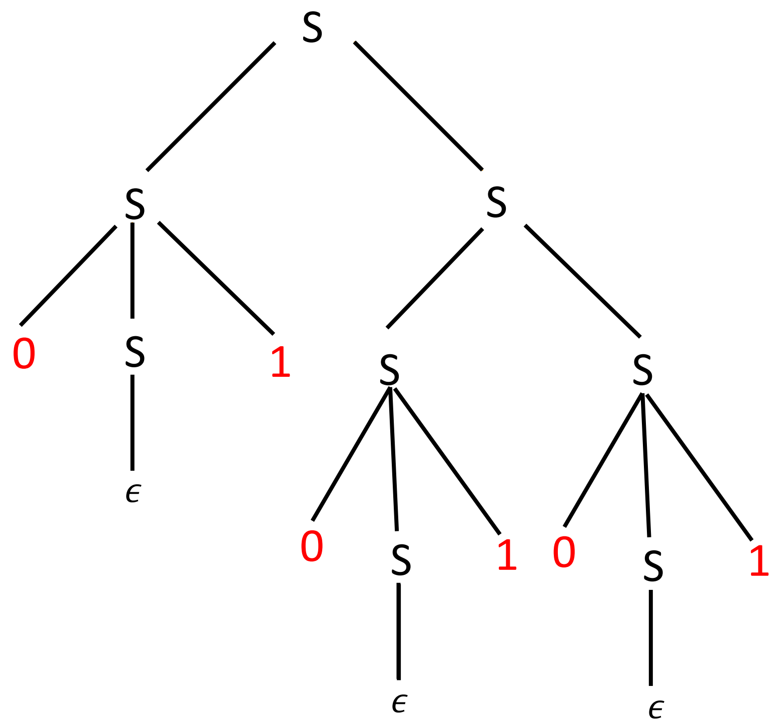
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \rightarrow 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{SS} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}S$

Parse trees for CFG

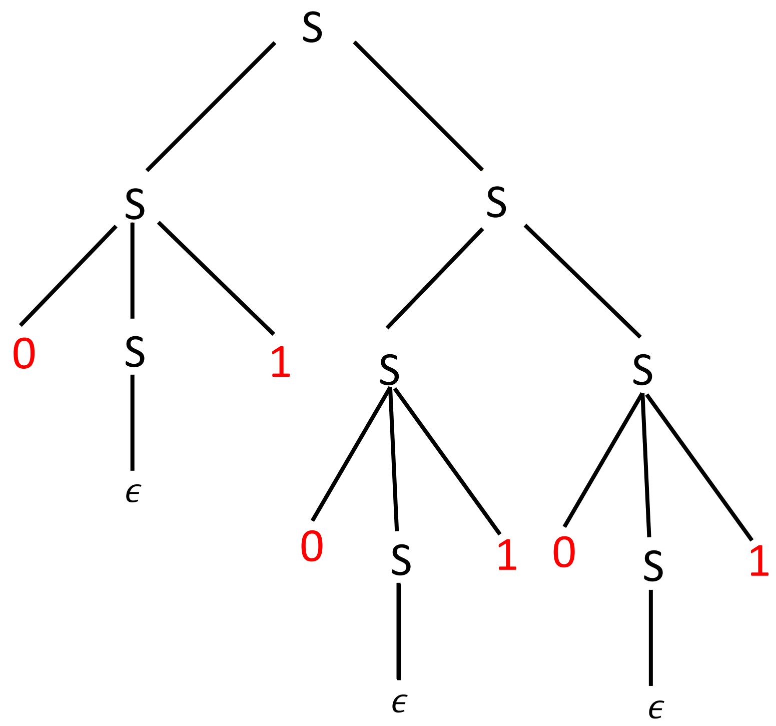
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}|\boldsymbol{SS}|\boldsymbol{\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{S}S \rightarrow 0\boldsymbol{S}1S \rightarrow 01S$

Parse trees for CFG

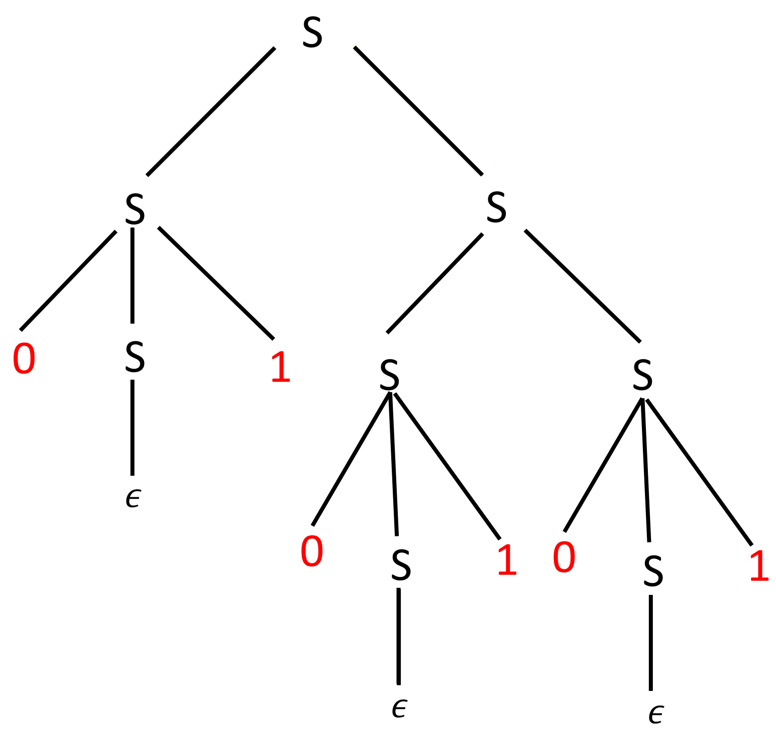
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \to 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\ \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega = 010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \to \boldsymbol{SS} \to 0\boldsymbol{S}1S \to 01\boldsymbol{S}$

Parse trees for CFG

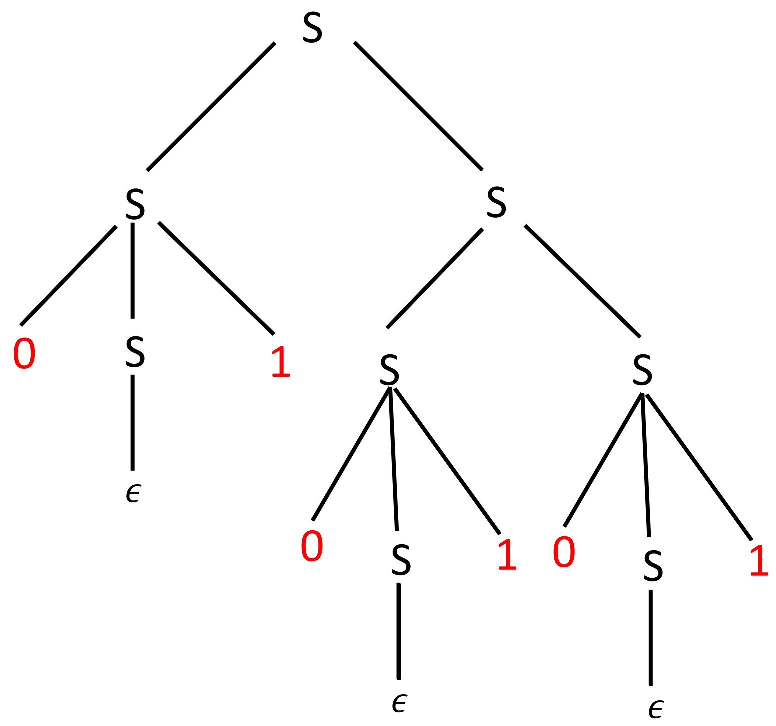
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}|\boldsymbol{SS}|\boldsymbol{\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{S}S \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{SS}$

Parse trees for CFG

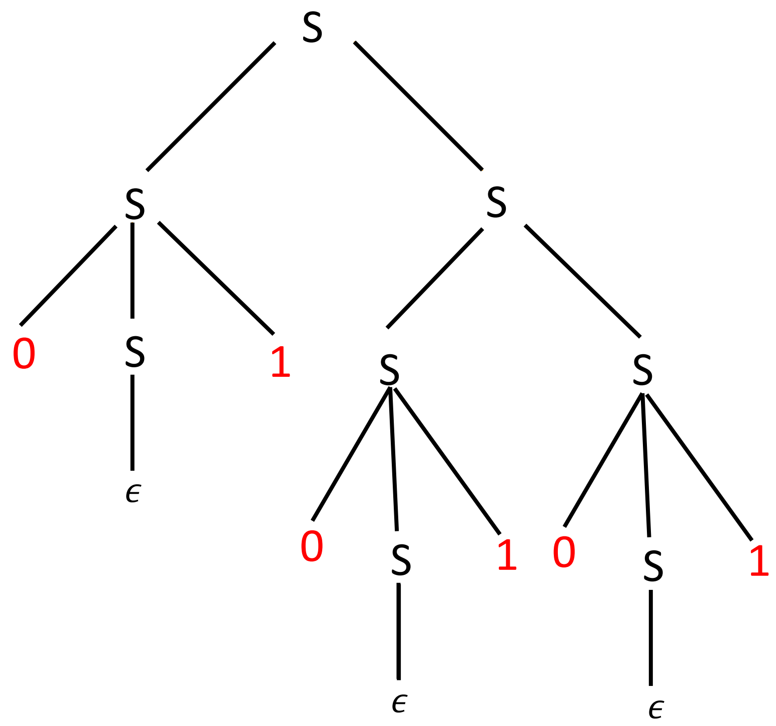
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \rightarrow 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega = 010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{S}S \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{S}S$

Parse trees for CFG

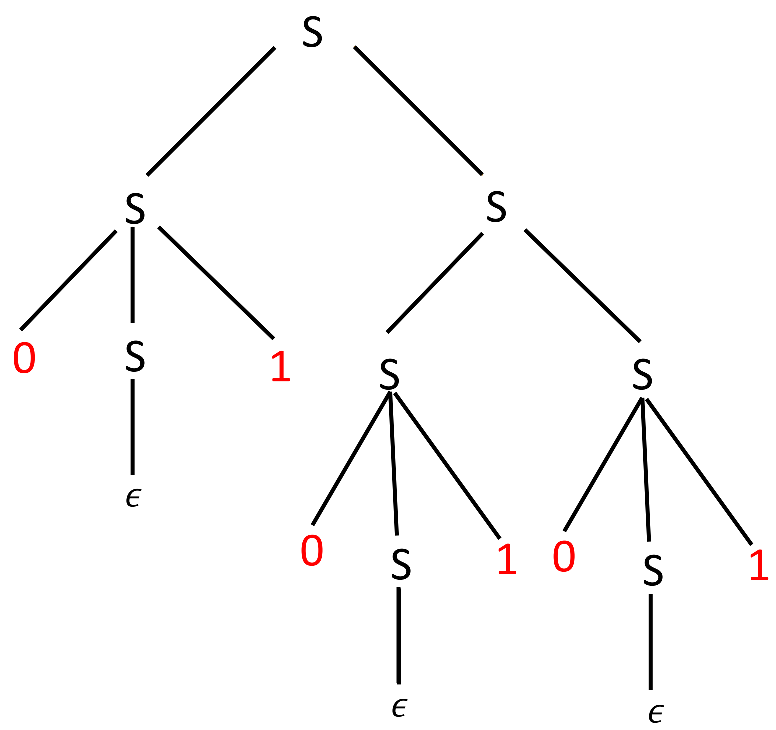
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \rightarrow 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega = 010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{SS} \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{SS} \rightarrow 01\boldsymbol{0S1}S$

Parse trees for CFG

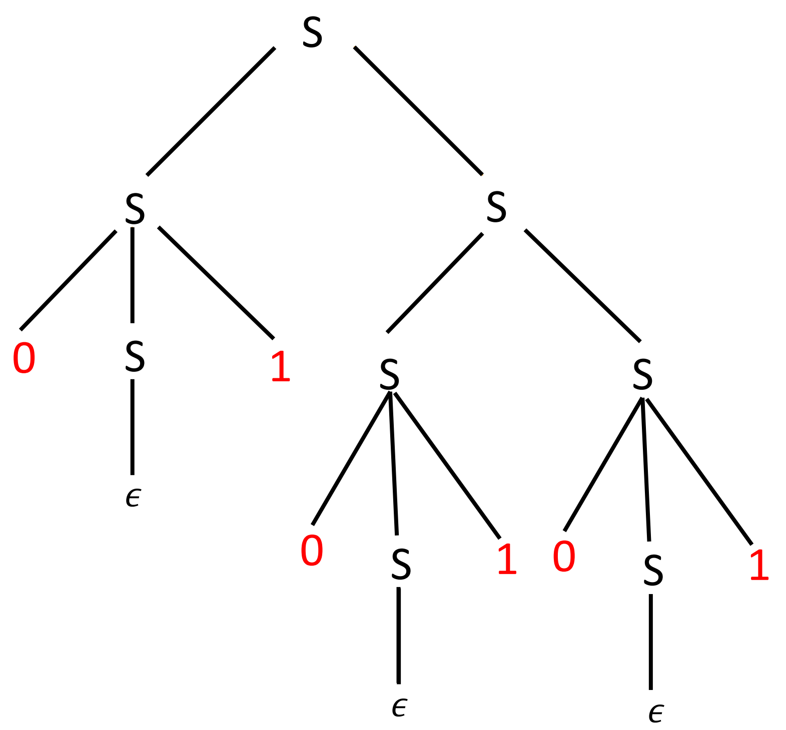
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for** $\boldsymbol{\omega}$ (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for** $\boldsymbol{\omega}$**.**

Consider the Grammar G with the following rules: $\boldsymbol{S \rightarrow 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega = 010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{S}S \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{S}S \rightarrow 010\boldsymbol{S}1S$

Parse trees for CFG

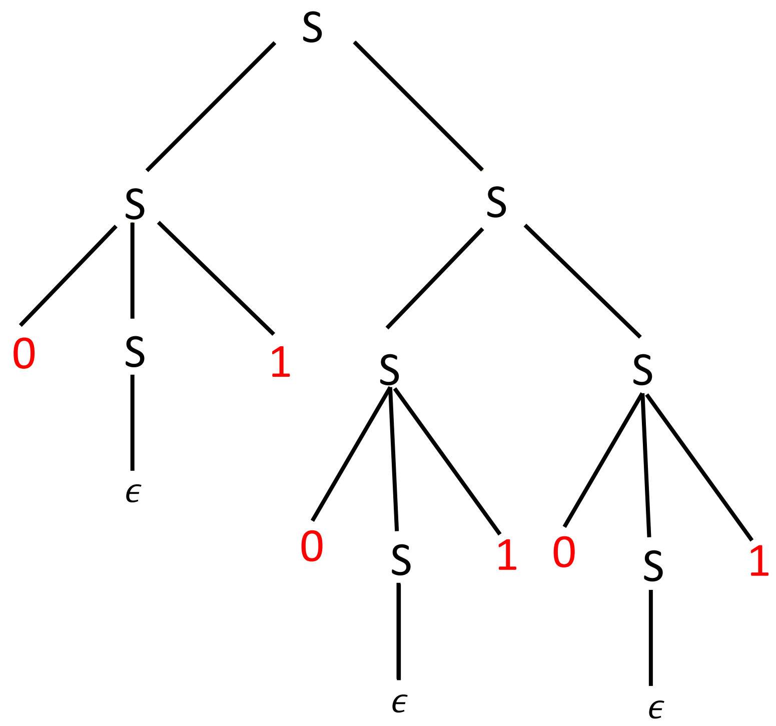
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \rightarrow 0S1|SS|\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega = 010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{S}S \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{S}S \rightarrow 010\boldsymbol{S}1S \rightarrow 0101S$

Parse trees for CFG

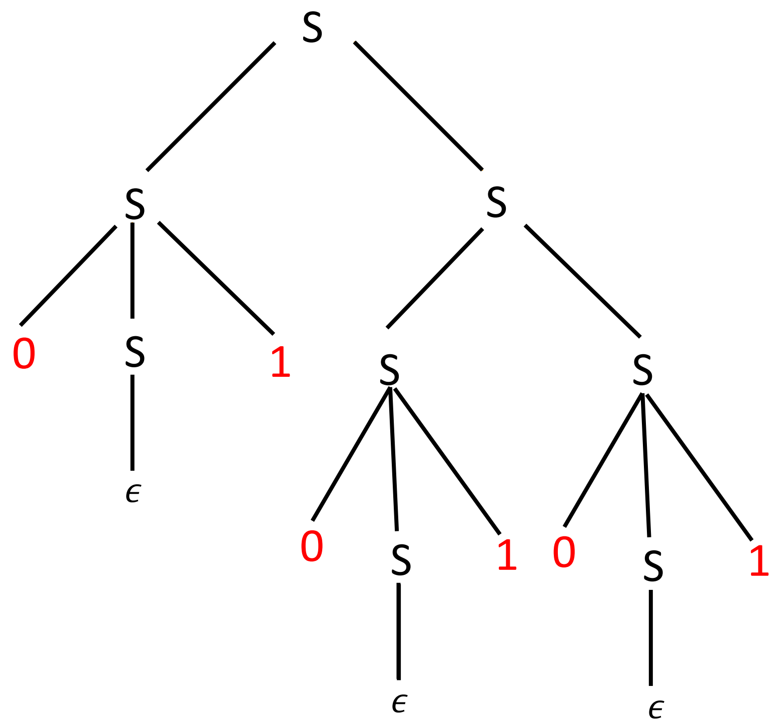
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}|\boldsymbol{SS}|\boldsymbol{\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{S}S \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{S}S \rightarrow 010\boldsymbol{S}1S \rightarrow 0101\boldsymbol{S}$

Parse trees for CFG

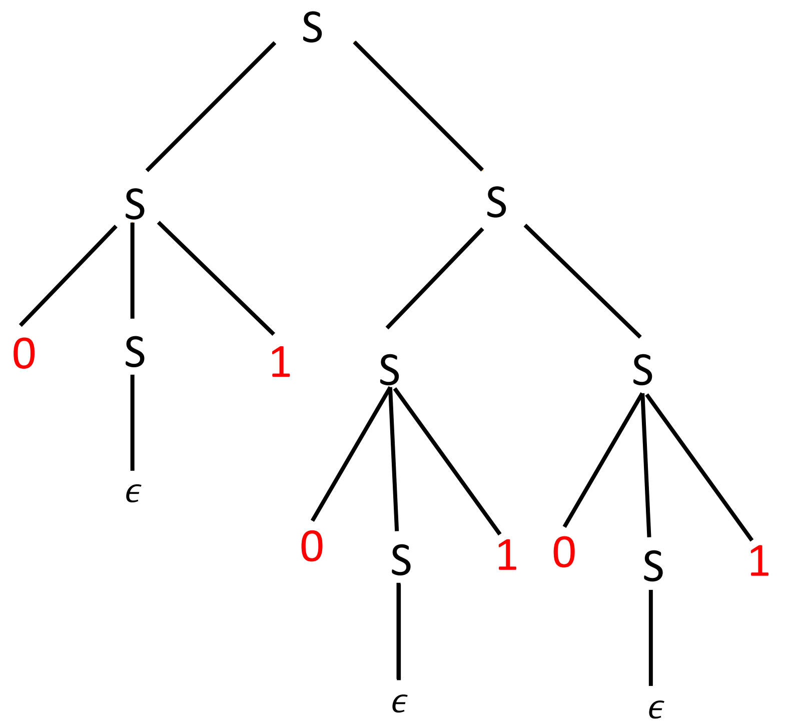
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}|\boldsymbol{SS}|\boldsymbol{\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{SS} \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{S}S \rightarrow 010\boldsymbol{S}1S \rightarrow 0101\boldsymbol{S} \rightarrow 0101\mathbf{0}\boldsymbol{S}\mathbf{1}$

Parse trees for CFG

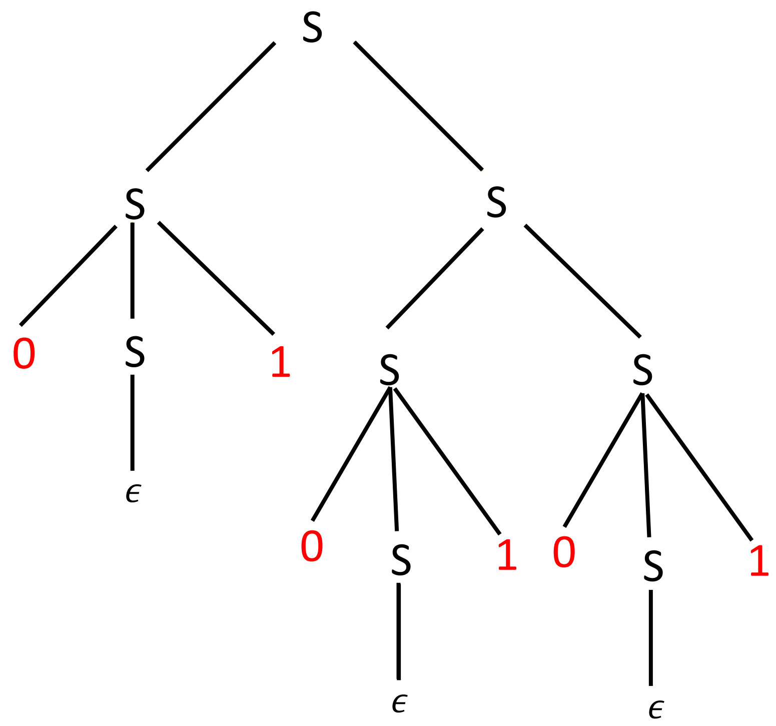
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}|\boldsymbol{SS}|\boldsymbol{\epsilon}$

Show that Grammar $\boldsymbol{G}$ is ambiguous, i.e. $\exists\, \omega \in L(G)$, such that there are two or more parse trees for ω.

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



Leftmost Derivation: $S \rightarrow \boldsymbol{SS} \rightarrow \mathbf{0}\boldsymbol{S}\mathbf{1}S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{SS} \rightarrow 01\mathbf{0}\boldsymbol{S}\mathbf{1}S \rightarrow 0101\boldsymbol{S} \rightarrow 0101\mathbf{0}\boldsymbol{S}\mathbf{1} \rightarrow \mathbf{010101}$

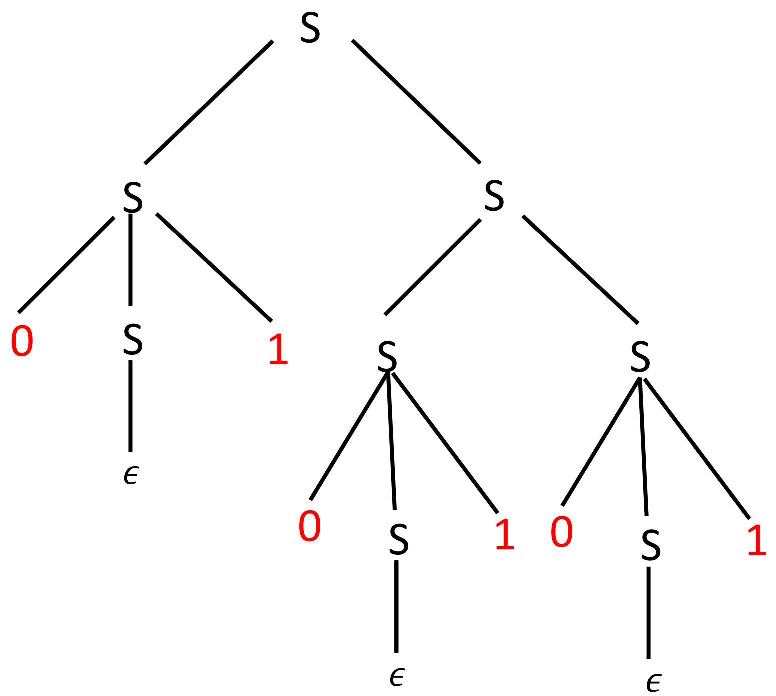
Parse trees for CFG

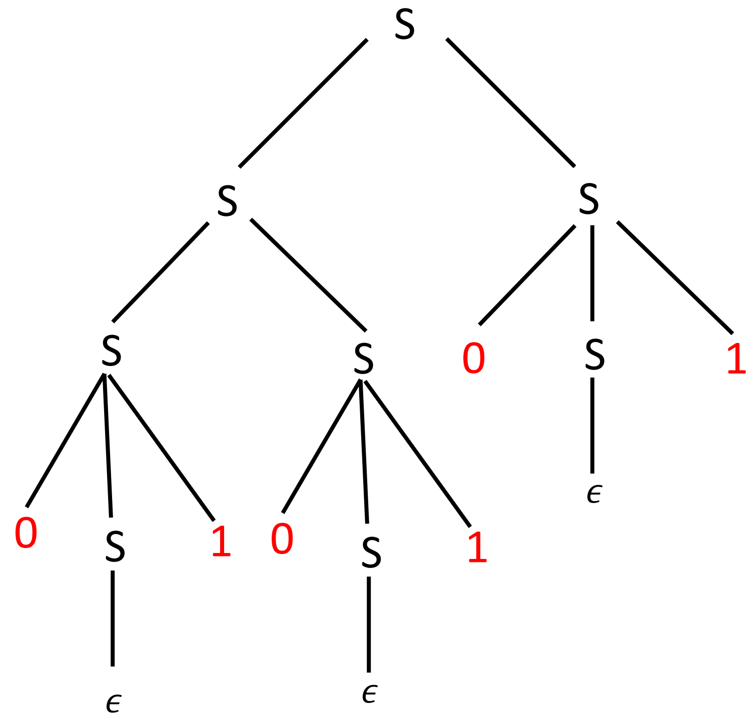
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$.**

Consider the Grammar G with the following rules: $\boldsymbol{S \to 0S1|SS|\epsilon}$

Consider the string $\boldsymbol{\omega} = \mathbf{010101}$:

- Show that there exist two different parse trees for **010101**.
- Show that there exist two leftmost derivations for **010101.**



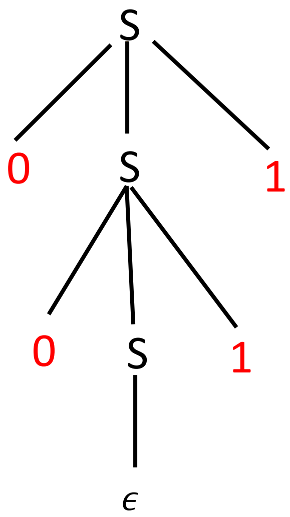


Leftmost Derivation: $S \to \boldsymbol{SS} \to \boldsymbol{SSS} \to \boldsymbol{0S1}SS \to 01SS \to 01\boldsymbol{0S1}S \to 0101\boldsymbol{S} \to 0101\boldsymbol{0S1} \to \mathbf{010101}$

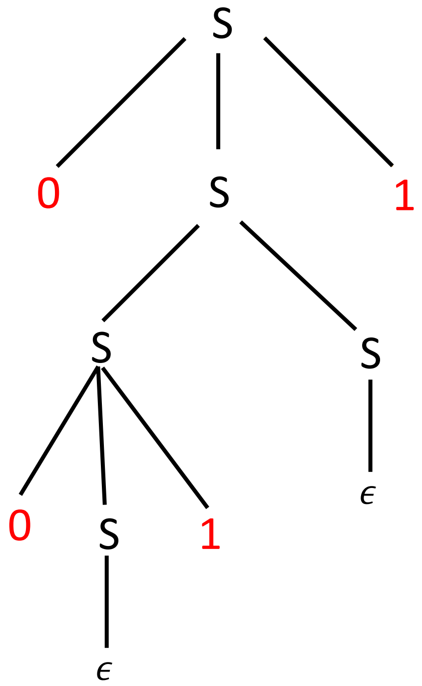
Parse trees for CFG

Show that the Grammar G with the following rules: $\boldsymbol{S \to 0S1 | SS | \epsilon}$ is ambiguous.

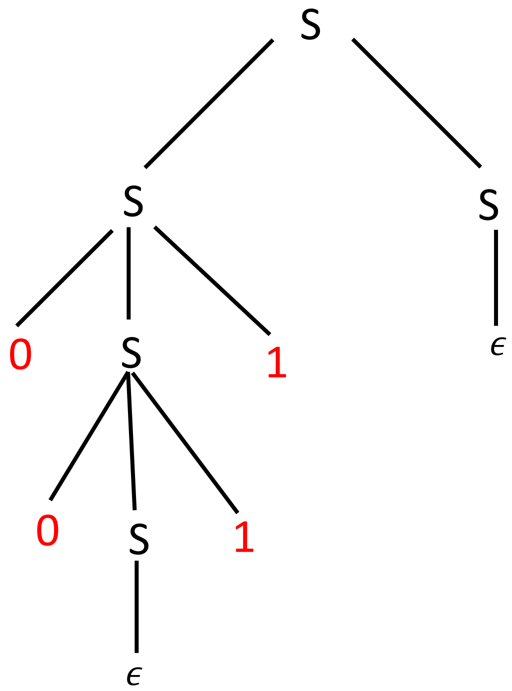
Consider string $\boldsymbol{\omega = 0011}$



LD: $S \to \mathbf{0}\boldsymbol{S}\mathbf{1} \to 0\mathbf{0}\boldsymbol{S}\mathbf{1}1 \to \mathbf{0011}$



LD: $S \to \mathbf{0}\boldsymbol{S}\mathbf{1} \to 0\boldsymbol{SS}1 \to 0\mathbf{0}\boldsymbol{S}\mathbf{1}S1 \to 001S1 \to \mathbf{0011}$



LD: $S \to \boldsymbol{SS} \to \mathbf{0}\boldsymbol{S}\mathbf{1}S \to 0\mathbf{0}\boldsymbol{S}\mathbf{1}1S \to 0011\boldsymbol{S} \to \mathbf{0011}$

Ambiguity

Unique structures are important. For example:

- The syntax of a programming language can be represented by a CFG.

- A compiler
  - translates the code written in the programming language into a form that is suitable for execution.
  - checks if the underlying programming language is syntactically correct.

- Parse trees are data structures that represent such structures.
- Parse tree for the code helps analyze the syntax. So ambiguity might lead to different interpretations and hence, different outcomes for the same code.

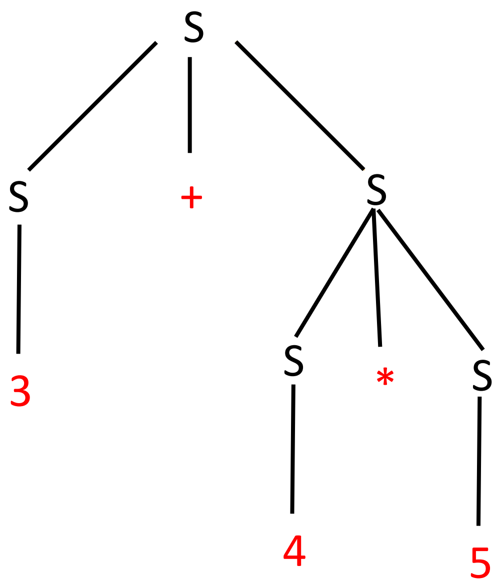
Ambiguity may not be desirable.

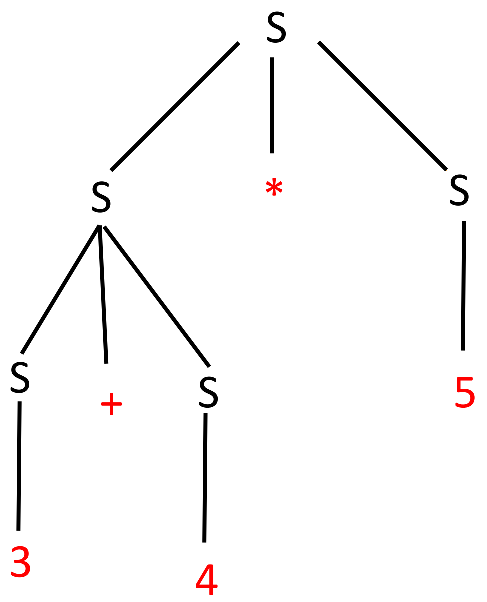
Ambiguity

Ambiguity may not be desirable.

Consider the grammar: $\boldsymbol{S \to S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

and the derivation of the string $\mathbf{3 + 4 * 5}$





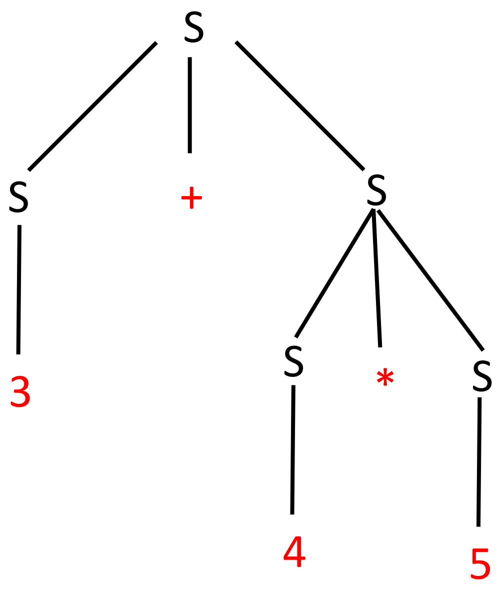
- The grammar contains no information on the precedence relations of the various arithmetic operations.
- The grammar may group $+$ before $*$

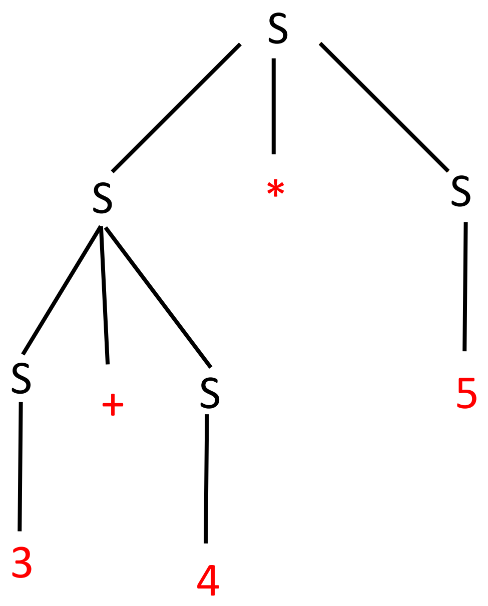
Ambiguity

Ambiguity may not be desirable.

Consider the grammar: $\boldsymbol{S \rightarrow S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

and the derivation of the string $\mathbf{3 + 4 * 5}$





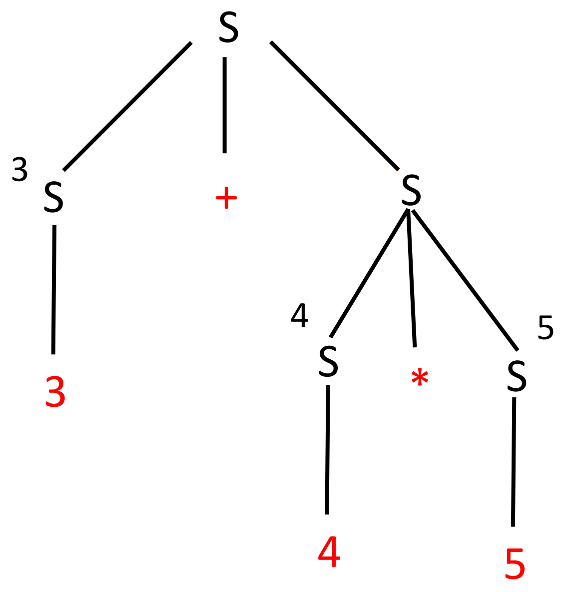
- What will be the result obtained from each of these ***parsings***?

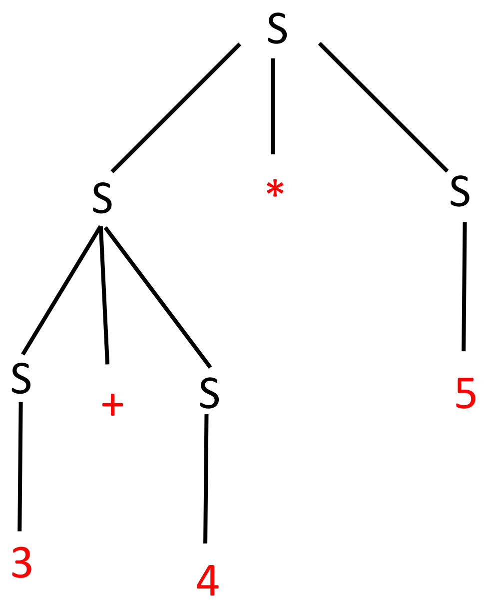
Ambiguity

Ambiguity may not be desirable.

Consider the grammar: $\boldsymbol{S \to S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

and the derivation of the string $\mathbf{3 + 4 * 5}$





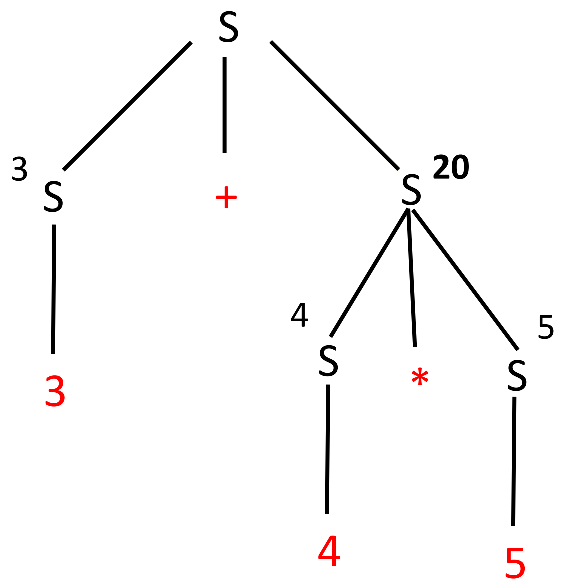
- If the compiler compiles the left parse tree

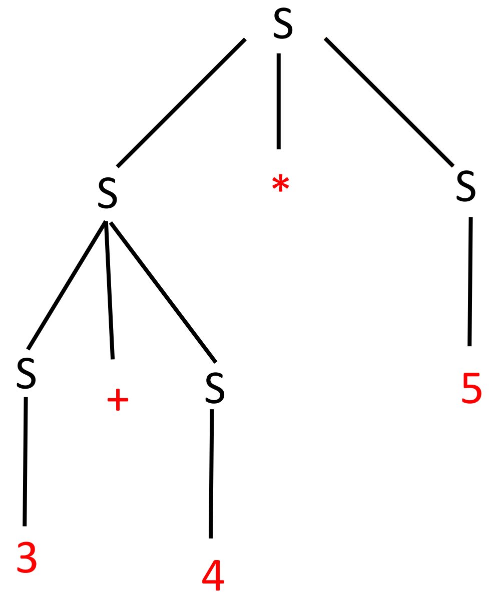
Ambiguity

Ambiguity may not be desirable.

Consider the grammar: $\boldsymbol{S \rightarrow S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

and the derivation of the string $\mathbf{3 + 4 * 5}$





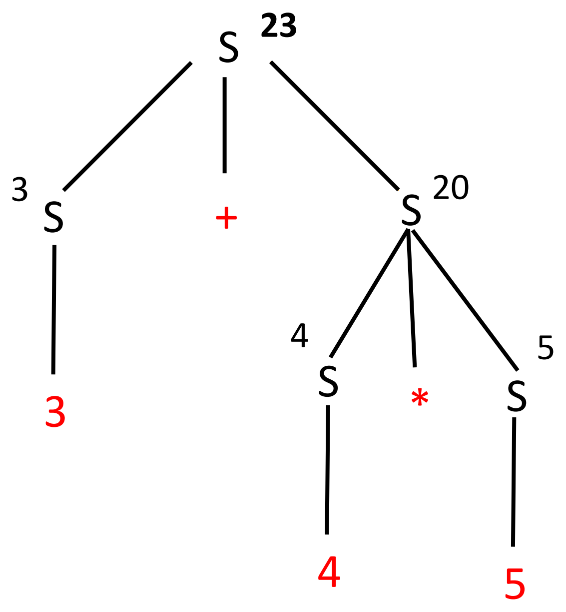
- If the compiler compiles the left parse tree

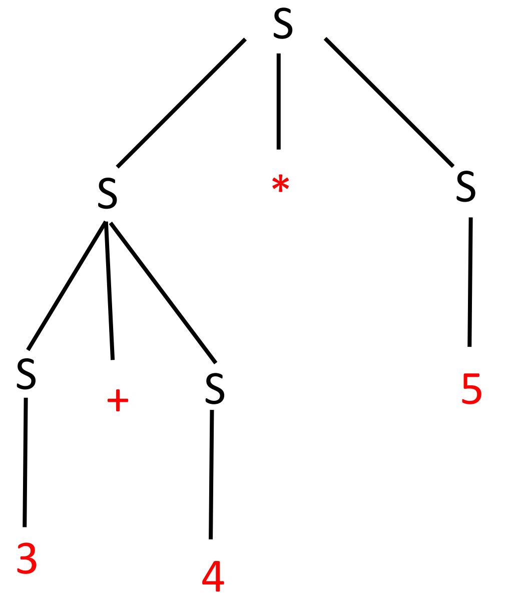
Ambiguity

Ambiguity may not be desirable.

Consider the grammar: $\boldsymbol{S \rightarrow S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

and the derivation of the string $\mathbf{3 + 4 * 5}$





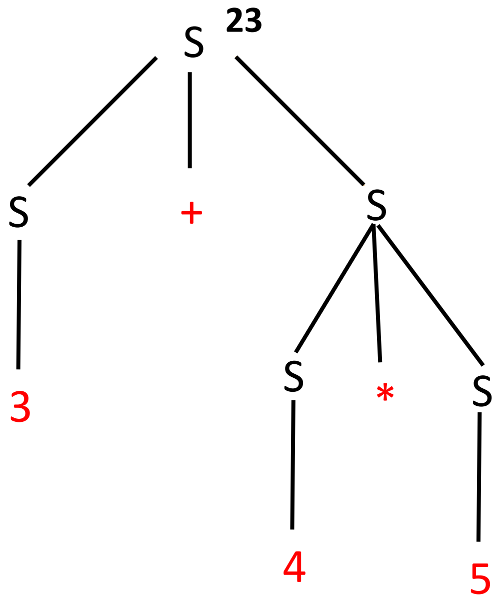
- If the compiler compiles the left parse tree. Outcome $= \mathbf{23}$

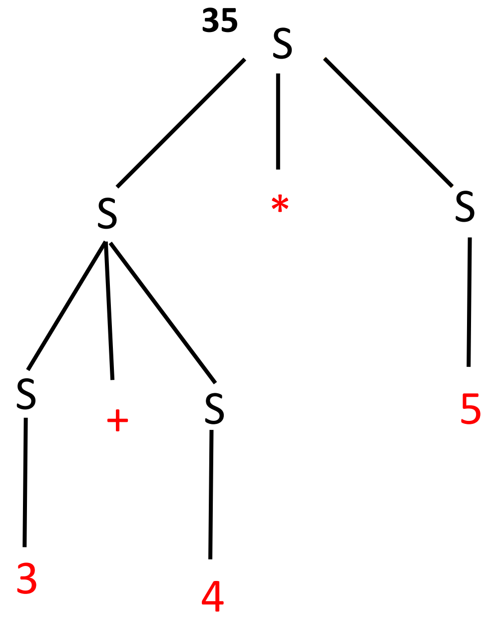
Ambiguity

Ambiguity may not be desirable.

Consider the grammar: $\boldsymbol{S \to S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

and the derivation of the string $\mathbf{3 + 4 * 5}$





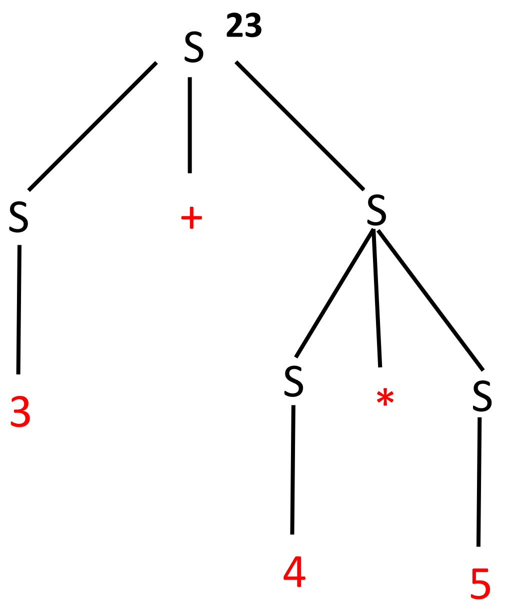
- If the compiler compiles the **right** parse tree. Outcome $= \mathbf{35}$

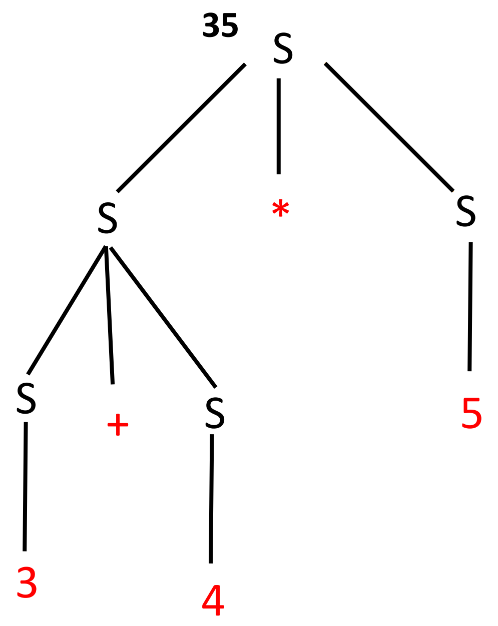
Ambiguity

Ambiguity may not be desirable.

Consider the grammar: $\boldsymbol{S} \rightarrow \boldsymbol{S} + \boldsymbol{S} \mid \boldsymbol{S} * \boldsymbol{S} \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \cdots \mid \mathbf{9}$

and the derivation of the string $\mathbf{3} + \mathbf{4} * \mathbf{5}$





- **How can we get rid of this ambiguity?**

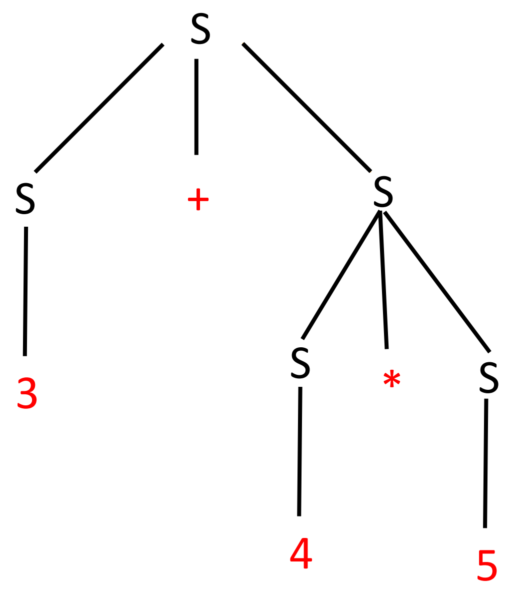
Ambiguity

Consider the grammar: $\boldsymbol{S} \rightarrow \boldsymbol{S} + \boldsymbol{S} \mid \boldsymbol{S} * \boldsymbol{S} \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \cdots \mid \mathbf{9}$

How can we get rid of this ambiguity? Change the production rules

1) Add parenthesis

New Grammar: $\boldsymbol{S} \rightarrow (\boldsymbol{S} + \boldsymbol{S}) \mid (\boldsymbol{S} * \boldsymbol{S}) \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \cdots \mid \mathbf{9}$



Old Parse tree (before adding parenthesis)

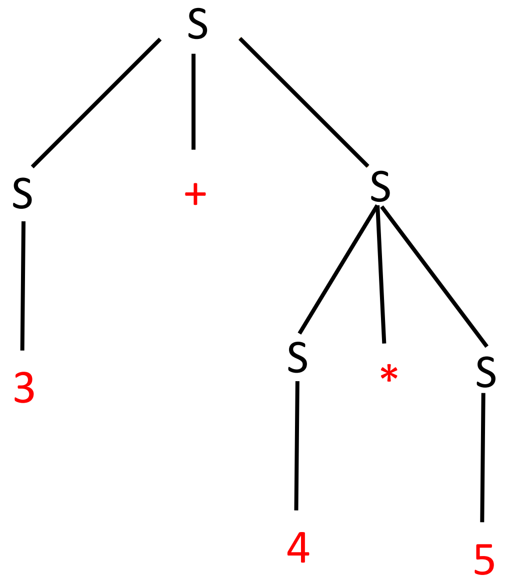
Ambiguity

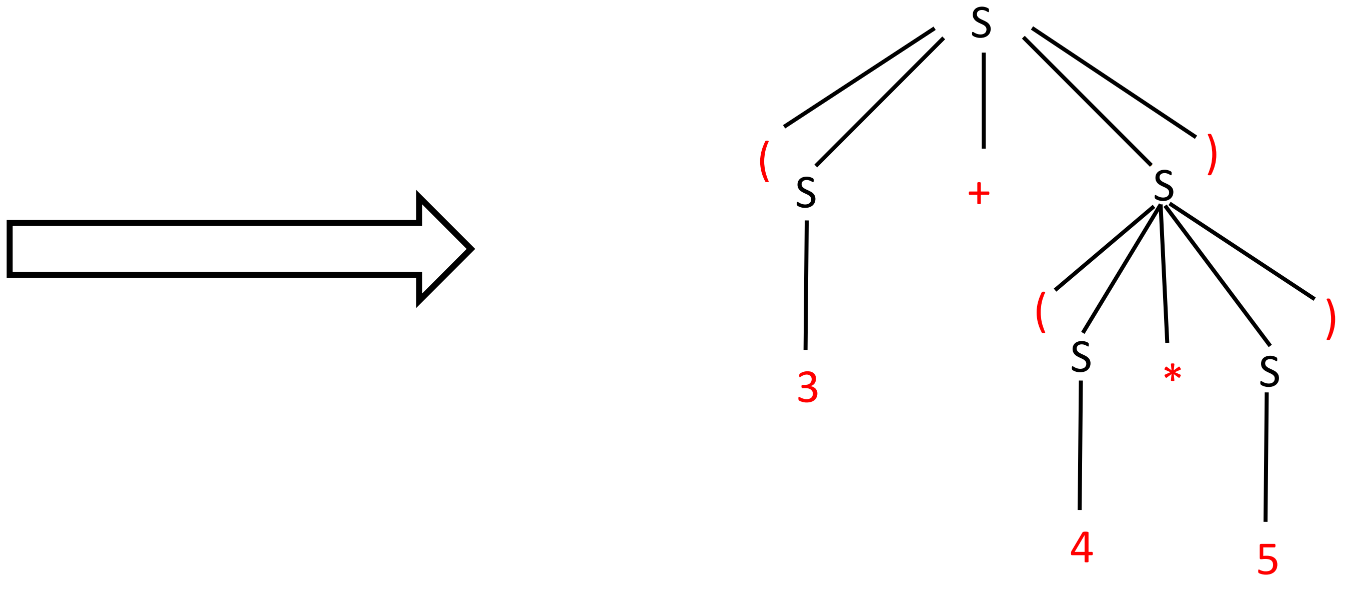
Consider the grammar: $\boldsymbol{S} \to \boldsymbol{S} + \boldsymbol{S} \mid \boldsymbol{S} * \boldsymbol{S} \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \cdots \mid \mathbf{9}$

How can we get rid of this ambiguity? Change the production rules

1) Add parenthesis

New Grammar: $\boldsymbol{S} \to (\boldsymbol{S} + \boldsymbol{S}) \mid (\boldsymbol{S} * \boldsymbol{S}) \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \cdots \mid \mathbf{9}$





$\big(3 + (4 * 5)\big) = 23$

Ambiguity

Consider the grammar: $\boldsymbol{S \rightarrow S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

How can we get rid of this ambiguity? Change the production rules

1) **Add parentheses**
2) **Add new variables**

Ambiguity

Consider the grammar: $\boldsymbol{S \rightarrow S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9}$

How can we get rid of this ambiguity? Change the production rules

1) **Add parentheses**
2) **Add new variables**

New Grammar:

$$\boldsymbol{E \rightarrow E + T \mid T}$$
$$\boldsymbol{T \rightarrow T * F \mid F}$$
$$\boldsymbol{F \rightarrow 0 \mid 1 \mid 2 \mid \cdots \mid 9 \mid E}$$

Ambiguity

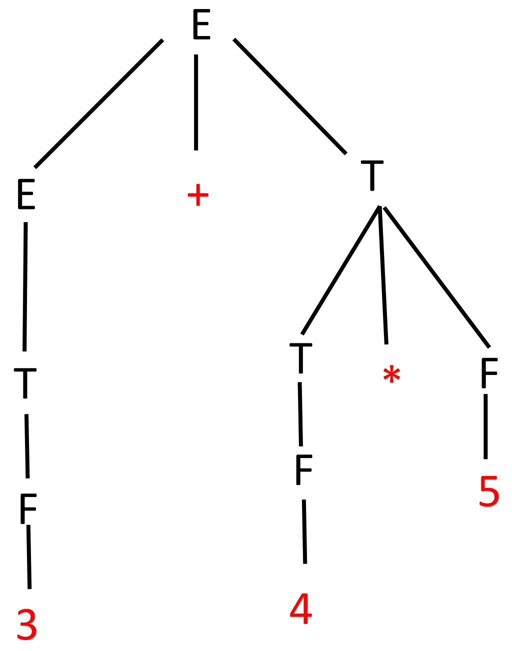
How can we get rid of this ambiguity? Change the production rules

1) **Add parentheses**
2) **Add new variables**

New Grammar:

$$\boldsymbol{E \to E + T \mid T}$$
$$\boldsymbol{T \to T * F \mid F}$$
$$\boldsymbol{F \to 0 \mid 1 \mid 2 \mid \cdots \mid 9 \mid E}$$

Parse tree to derive: $\mathbf{3 + (4 * 5)}$



Ambiguity

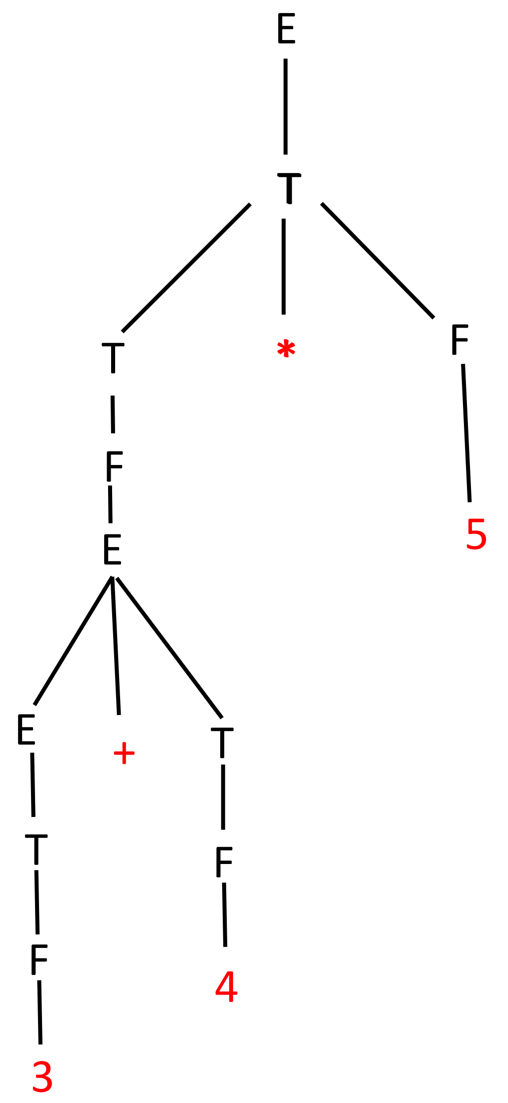
How can we get rid of this ambiguity? Change the production rules

1) **Add parentheses**
2) **Add new variables**

New Grammar:

$$\boldsymbol{E \rightarrow E + T \mid T}$$
$$\boldsymbol{T \rightarrow T * F \mid F}$$
$$\boldsymbol{F \rightarrow 0 \mid 1 \mid 2 \mid \cdots \mid 9 \mid E}$$

Parse tree to derive: $\boldsymbol{(3 + 4) * 5}$



Ambiguity

How can we get rid of this ambiguity? Change the production rules

1) **Add parentheses**
2) **Add new variables**

- In general, it is not possible to write an algorithm that takes as input a grammar G and outputs, YES if G is ambiguous and NO, otherwise. **(Undecidable)**

- So removing ambiguity is impossible in general.

Chomsky Normal Form

Often it is easier to work with CFG in a simple standardized form - the Chomsky Normal Form (CNF) is one of them.

Chomsky Normal Form

A CFG G is in CNF if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\,Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\,Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\,Var$.

Chomsky Normal Form

Often it is easier to work with CFG in a simple standardized form - the Chomsky Normal Form (CNF) is one of them.

Chomsky Normal Form

A CFG G is in CNF if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\,Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\,Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\,Var$.

Why are CNFs useful?

- Suppose you are given a CFG G and a string w as input and you have to write an algorithm that decides whether G generates w.
- Your algorithm outputs YES if G generates w and NO, otherwise.

Chomsky Normal Form

A CFG G is in **CNF** if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\,Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\,Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\ Var$.

Why are CNFs useful?

- Suppose you are given a CFG G as and a string w as input and you have to **write an algorithm that decides whether $\boldsymbol{G}$ generates $\boldsymbol{w}$**.

- The algorithm outputs YES if G generates w and NO, otherwise.

❖ One idea is to go through ALL derivations one by one and output YES if any of them generates w.
❖ However, infinitely many derivations may have to tried.
❖ So if G does not generate w, the algorithm will never stop.
❖ So this problem appears to be **undecidable**.

Chomsky Normal Form

A CFG G is in **CNF** if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\,Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\,Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\,Var$.

Why are CNFs useful?

Suppose you are given a CFG G and a string w as input and you have to **write an algorithm that decides whether $\boldsymbol{G}$ generates $\boldsymbol{w}$**. This problem appears to be **undecidable**.

Chomsky Normal Form

A CFG G is in **CNF** if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\,Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\,Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\,Var$.

Why are CNFs useful?

Suppose you are given a CFG G as and a string w as input and you have to **write an algorithm that decides whether $\boldsymbol{G}$ generates $\boldsymbol{w}$**.

- Converting G first to a CNF alleviates this and **makes the problem decidable**.
- It limits the number of steps in derivations required to generate any $w \in L(G)$.
- If $w \in L(G)$, then a CFG in Chomsky Normal Form has **derivations of $\boldsymbol{2n-1}$ steps** for input strings w of length n (We will prove this shortly).

Chomsky Normal Form

A CFG G is in **CNF** if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\, Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\, Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\ Var$.

A CFG in Chomsky Normal Form has derivations of $2n - 1$ steps for generating strings $w \in L(G)$ of length n.

Why are CNFs useful?

Suppose you are given a CFG G as and a string w as input and you have to **write an algorithm that decides whether $\boldsymbol{G}$ generates $\boldsymbol{w}$**.

1. Convert G to CNF.
2. List all derivations of $2n - 1$ steps, where $|w| = n$. (There are a finite number of these)
3. If ANY of these derivations generate w, output YES, otherwise output NO.

Chomsky Normal Form

A CFG G is in **CNF** if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\,Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\,Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\ Var$.

1) **A CFG in Chomsky Normal Form has derivations of $\boldsymbol{2n-1}$ steps for generating strings $\boldsymbol{w \in L(G)}$ of length $\boldsymbol{n}$.**

2) Any CFL can be generated by a CFG written in Chomsky Normal Form.

To prove 1) use induction!

Chomsky Normal Form

Prove that **a CFG in Chomsky Normal Form has derivations of $\mathbf{2n-1}$ steps for generating strings $\boldsymbol{w} \in \boldsymbol{L(G)}$ of length $\boldsymbol{n}$.**

Proof: Note that any CFG in CNF can be written as:

$$\boldsymbol{A \to BC} \qquad [B, C \text{ are not start variables}]$$
$$\boldsymbol{A \to a} \qquad [a \text{ is a terminal}]$$
$$\boldsymbol{S \to \epsilon} \qquad [S \text{ is the Start Variable}]$$

We will prove this by **induction.**

(Basic step) Let $|w| = 1$. Then **one** application of the second rule would suffice. So any derivation of w would need $2|w| - 1 = 1$ step.
(Inductive hypothesis) Assume the statement of the theorem to be true for any string of length at most k where $k \geq 1$. Now we shall show that it holds for any $w \in L(G)$ such that $|w| = k + 1$.

Chomsky Normal Form

Prove that **a CFG in Chomsky Normal Form has derivations of $2n - 1$ steps for generating strings $w \in L(G)$ of length n.**

Proof: Note that any CFG in CNF can be written as:

$$\begin{array}{ll} \boldsymbol{A \rightarrow BC} & [B, C \text{ are not start variables}] \\ \boldsymbol{A \rightarrow a} & [a \text{ is a terminal}] \\ \boldsymbol{S \rightarrow \epsilon} & [S \text{ is the Start Variable}] \end{array}$$

We will prove this by **induction.**

(Basic step) Let $|w| = 1$. Then **one** application of the second rule would suffice. So any derivation of w would need $2|w| - 1 = 1$ step.
(Inductive hypothesis) Assume the statement of the theorem to be true for any string of length at most k where $k \geq 1$. Now we shall show that it holds for any $w \in L(G)$ such that $|w| = k + 1$.

Since $|w| > 1$, any derivation will start from the rule $A \rightarrow BC$. So $\boldsymbol{w = xy}$**, where** $\boldsymbol{B \overset{*}{\Rightarrow} x, |x| > 0}$ **and** $\boldsymbol{C \overset{*}{\Rightarrow} y}, \boldsymbol{|y| > 0}$. But since $|x|, |y| \leq k$, and we have that by the inductive hypothesis: (i) number of steps in the derivation $\boldsymbol{B \overset{*}{\Rightarrow} x}$ **is** $\boldsymbol{2|x| - 1}$ and (ii) number of steps in the derivation $\boldsymbol{C \overset{*}{\Rightarrow} y}$ **is** $\boldsymbol{2|y| - 1}$. So the number of steps in the derivation of w is

$$\boldsymbol{1 + (2|x| - 1) + (2|y| - 1) = 2(|x| + |y|) - 1 = 2|w| - 1 = 2(k + 1) - 1}.$$

Chomsky Normal Form

A CFG G is in **CNF** if every rule of G is of the form

$$\boldsymbol{Var \rightarrow Var\, Var}$$
$$\boldsymbol{Var \rightarrow ter}$$
$$\boldsymbol{Start\, Var \rightarrow \epsilon}$$

where Var can be any variable, including the Start Variable, $Start\ Var$.

1) A CFG in Chomsky Normal Form has derivations of $2n - 1$ steps for generating strings $w \in L(G)$ of length n.

2) **Any CFL can be generated by a CFG written in Chomsky Normal Form.**

Chomsky Normal Form

Any CFL can be generated by a CFG written in Chomsky Normal Form.

Proof: The proof is constructive. Suppose we have a CFG G with a set of rules. To convert G into CNF, we do the following:

1. Add a new start variable $S' \to S$

2. Remove ϵ rules of the form $A \to \epsilon$
   - Remove nullable symbols/rules

3. Remove unit (short) rules of the form $A \to B$
   - Remove useless symbols/rules

4. Remove long rules of the form $A \to u_1 u_2 \cdots u_k$
   - Remove useless symbols/rules

Thank You!