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

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.**

- 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

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.**

- 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 $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$

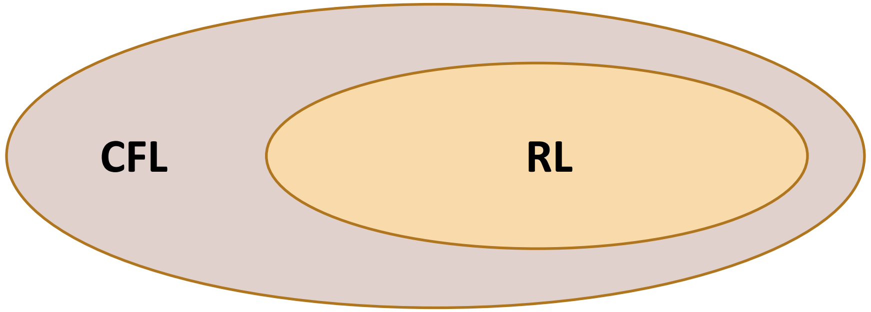
Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

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

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.



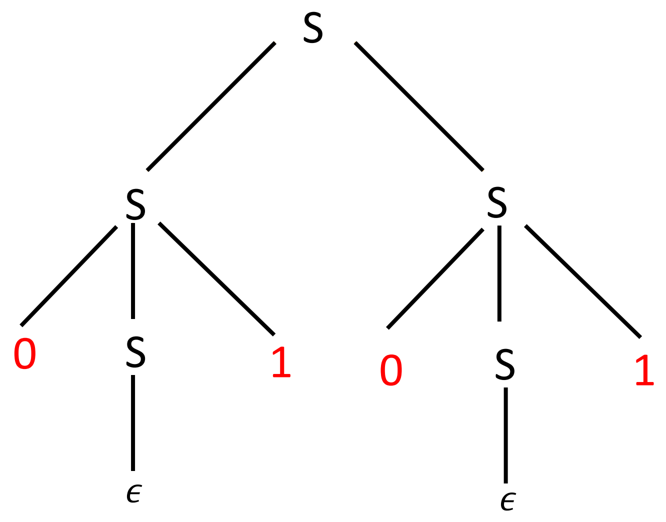
Parse trees for CFG

Consider the Grammar G with the following rules:

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

One derivation:

$$S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{0S1}S \rightarrow 0S1\boldsymbol{0S1} \rightarrow \mathbf{0101}$$



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 $\boldsymbol{S} \overset{*}{\Rightarrow} \boldsymbol{w}$ and $\boldsymbol{w} \in \boldsymbol{L(G)}$

Parse trees for CFG

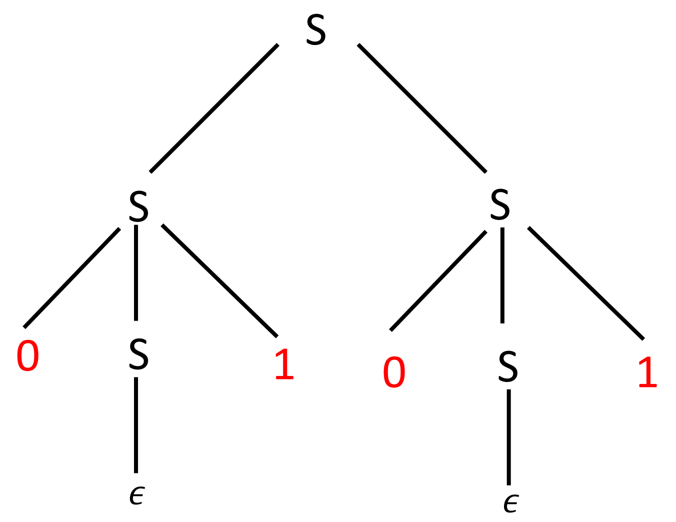
Consider the Grammar G with the following rules:

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

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

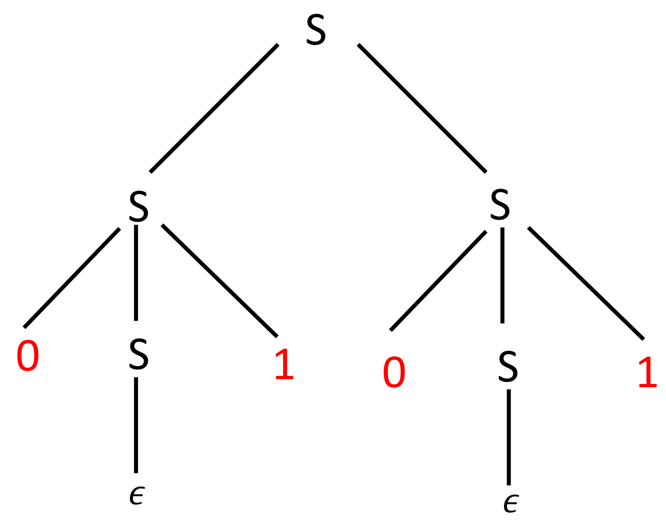
Consider the Grammar G with the following rules:

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

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

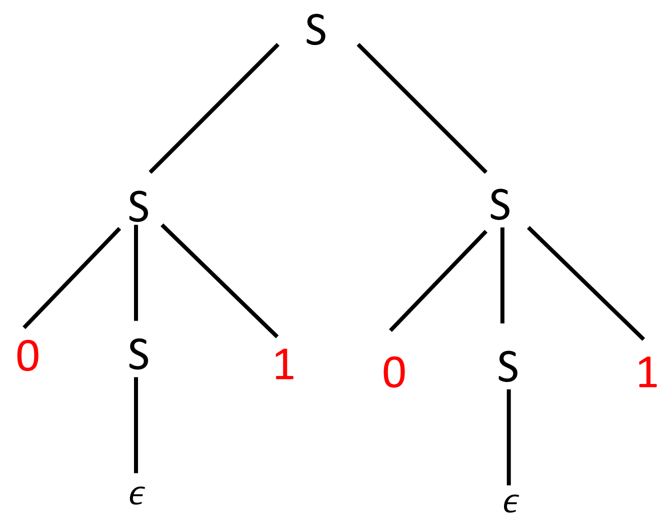
Consider the Grammar G with the following rules:

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

Consider the following derivations for 0101:

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

- 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 $\boldsymbol{\omega} \in \boldsymbol{L(G)}$**,** 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 $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** 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.**

Parse trees for CFG

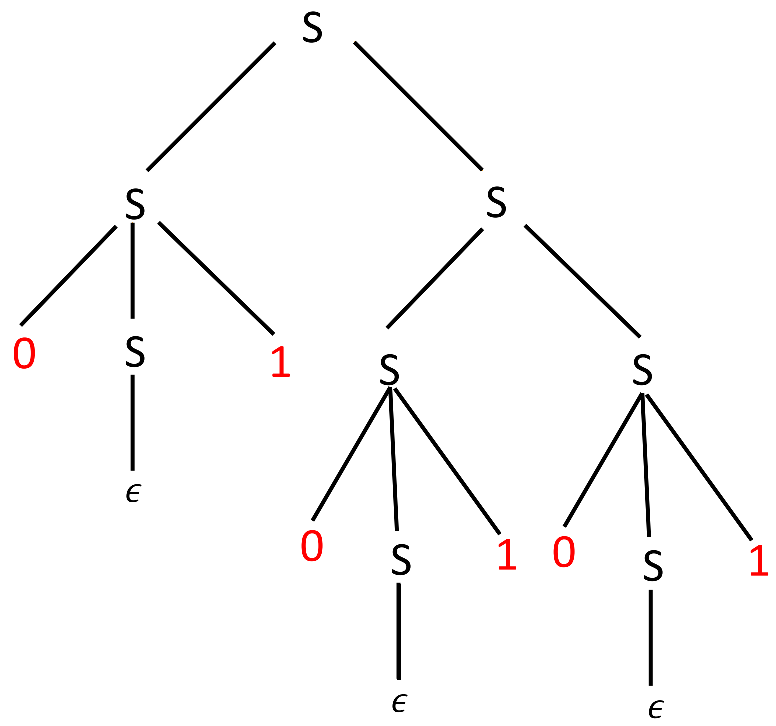
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for** $\boldsymbol{\omega}$ (or equivalently two or more rightmost derivations) or equivalently **two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

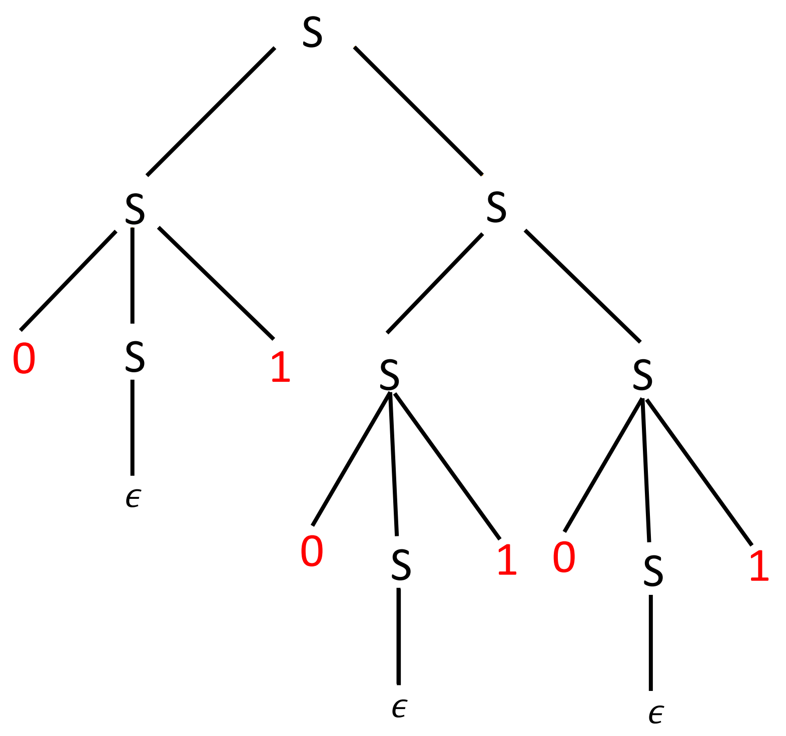
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

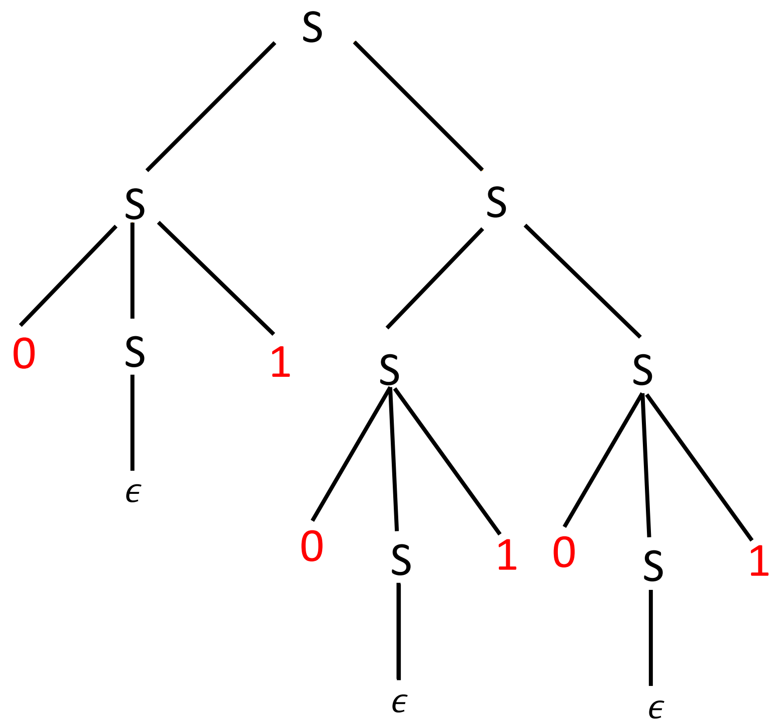
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for** $\boldsymbol{\omega}$ (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

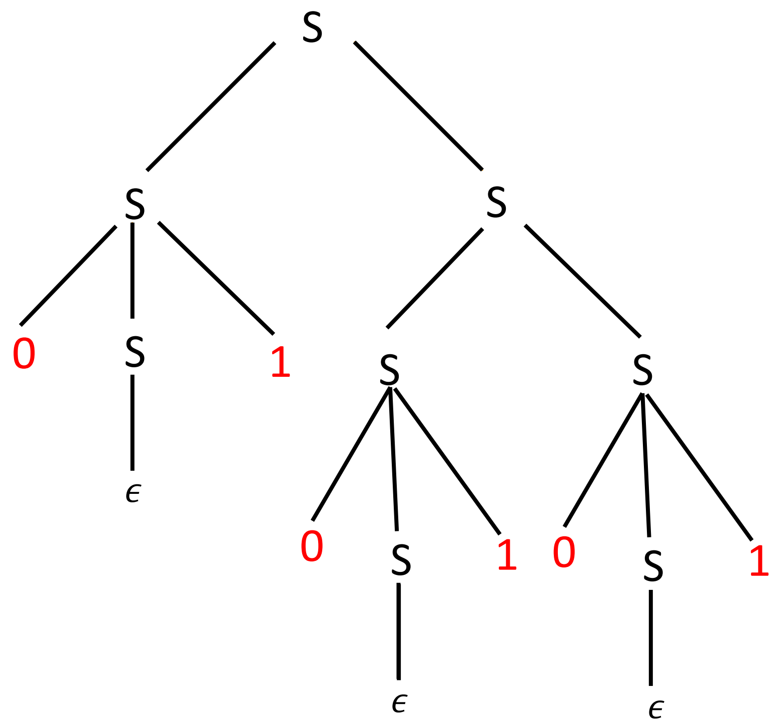
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for** $\boldsymbol{\omega}$ (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

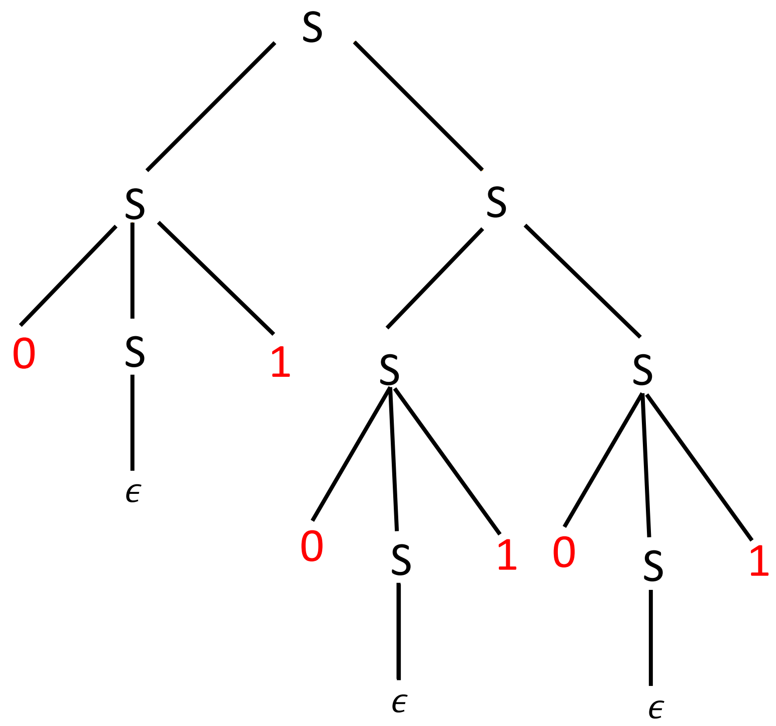
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

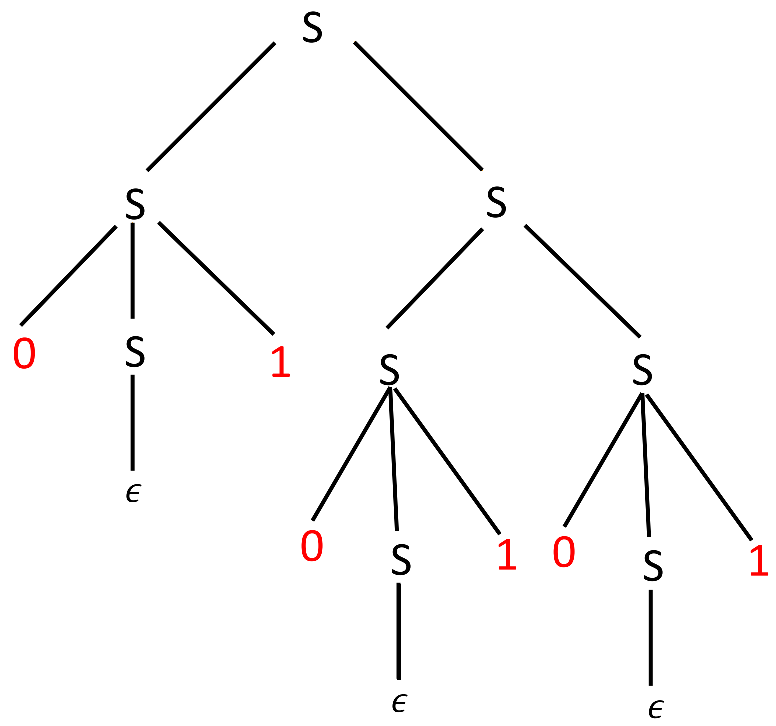
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

Parse trees for CFG

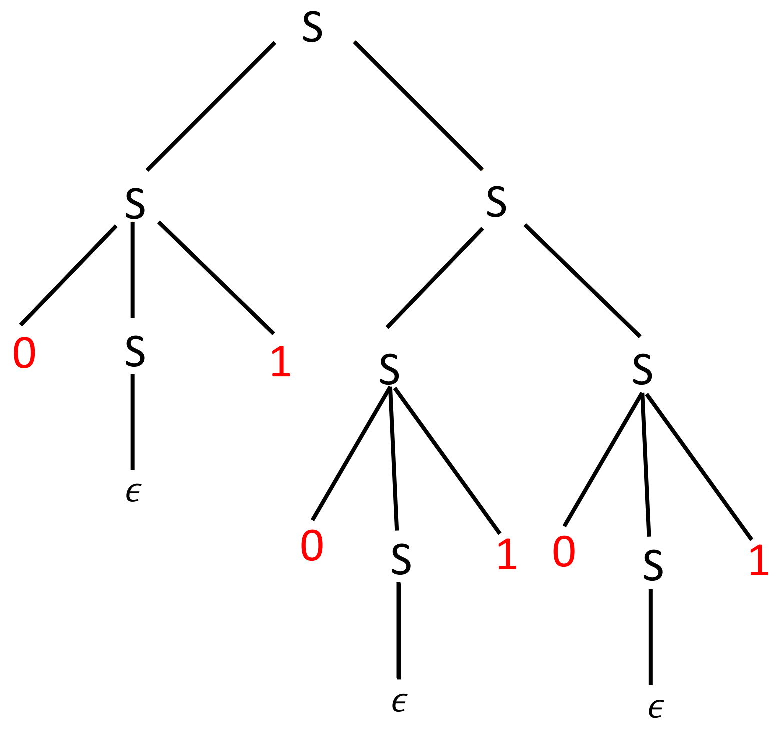
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

Parse trees for CFG

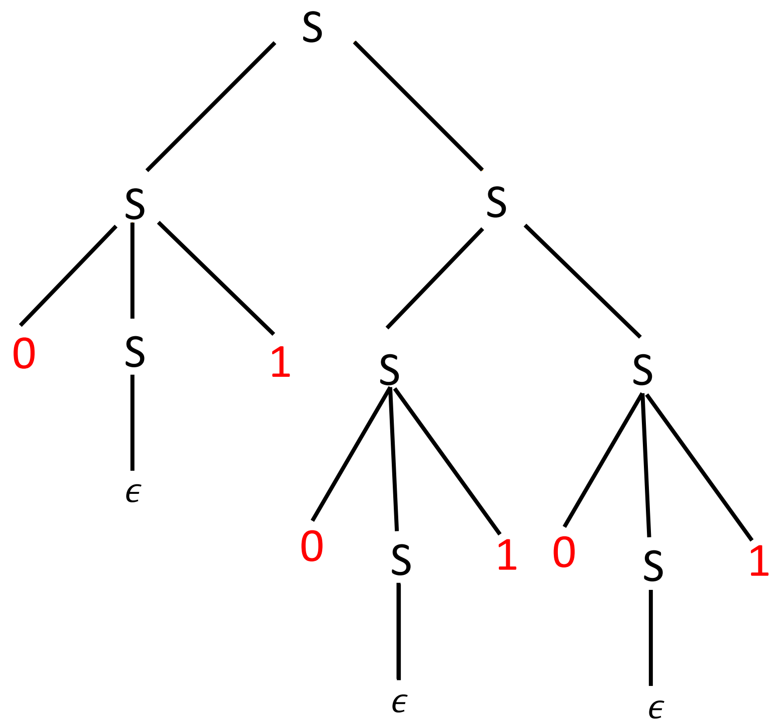
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for** $\boldsymbol{\omega}$ (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

Parse trees for CFG

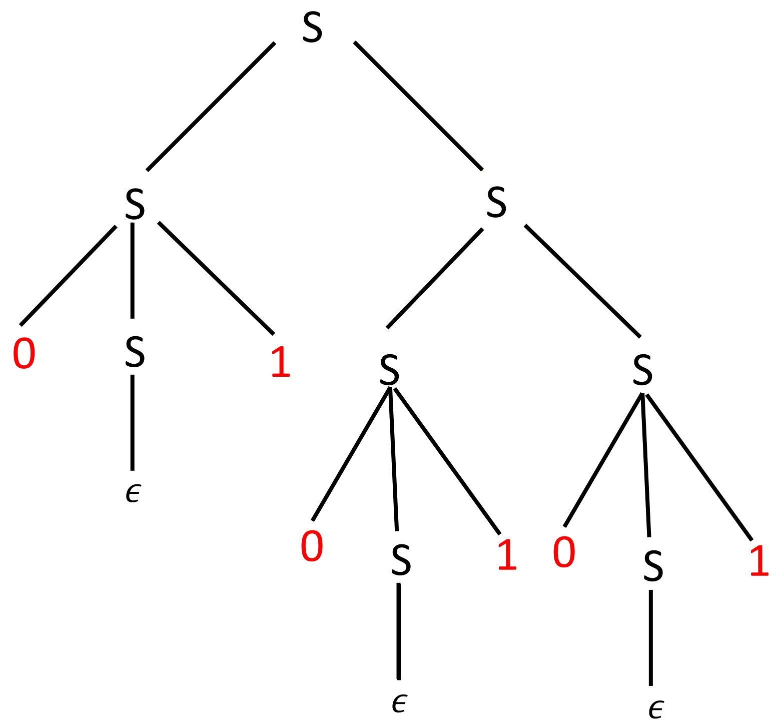
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two 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: $S \rightarrow \boldsymbol{S}S \rightarrow 0\boldsymbol{S}1S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{S}S \rightarrow 01\mathbf{0}\boldsymbol{S}\mathbf{1}S$

Parse trees for CFG

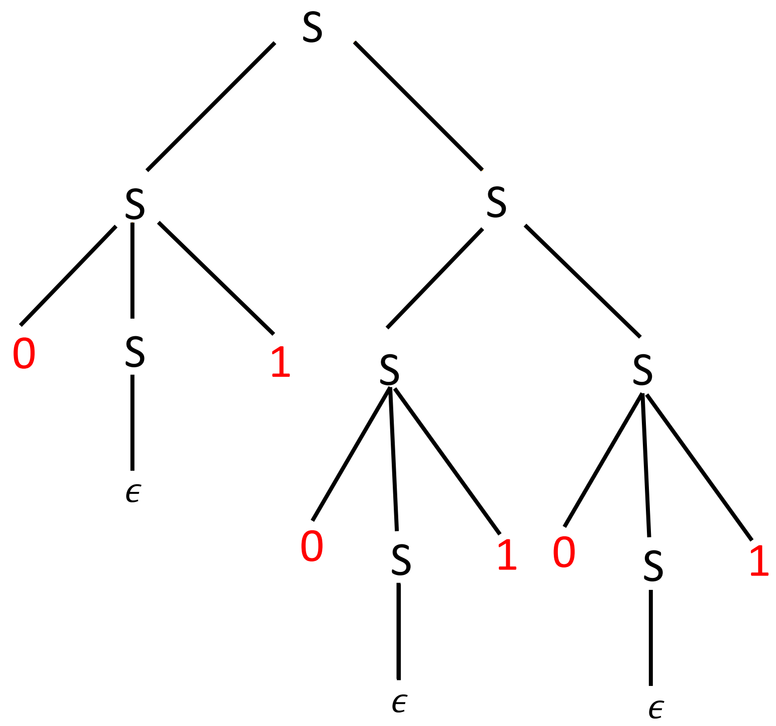
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two 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 $\boldsymbol{\omega = 010101}$:

- 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

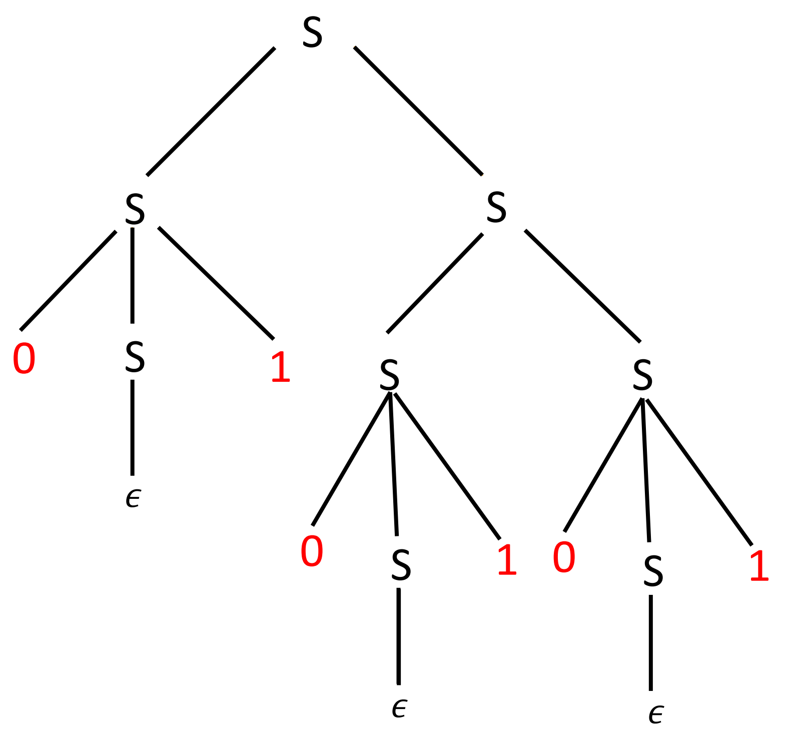
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

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

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

Parse trees for CFG

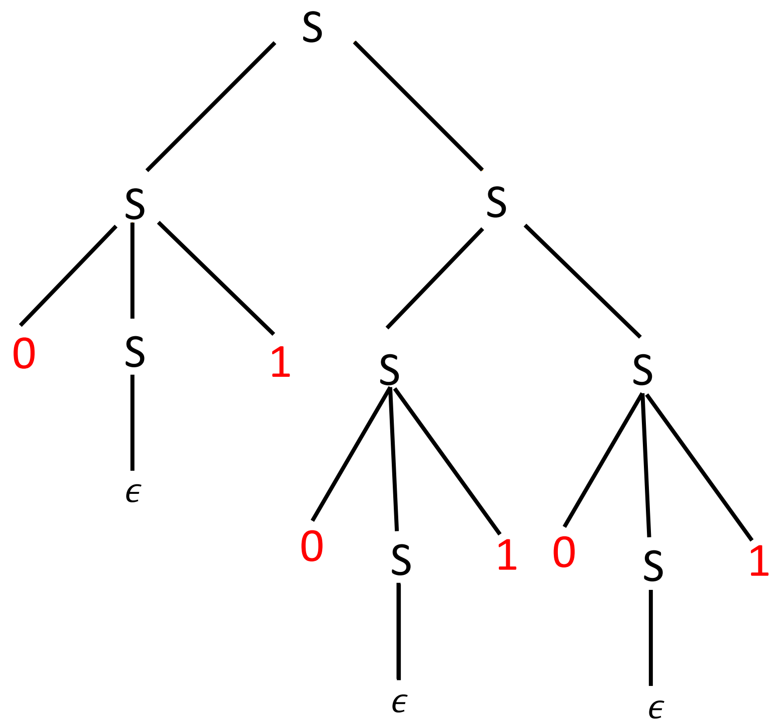
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

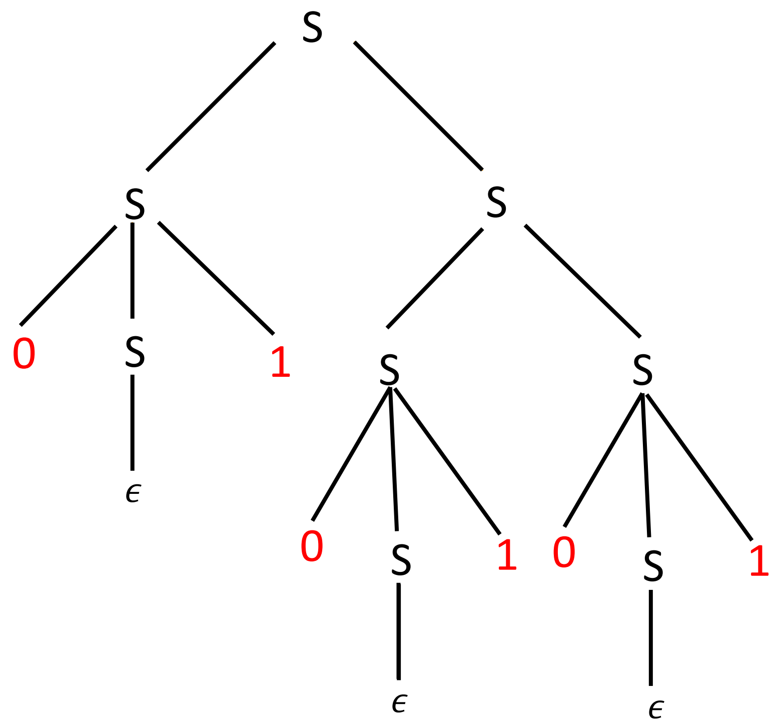
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

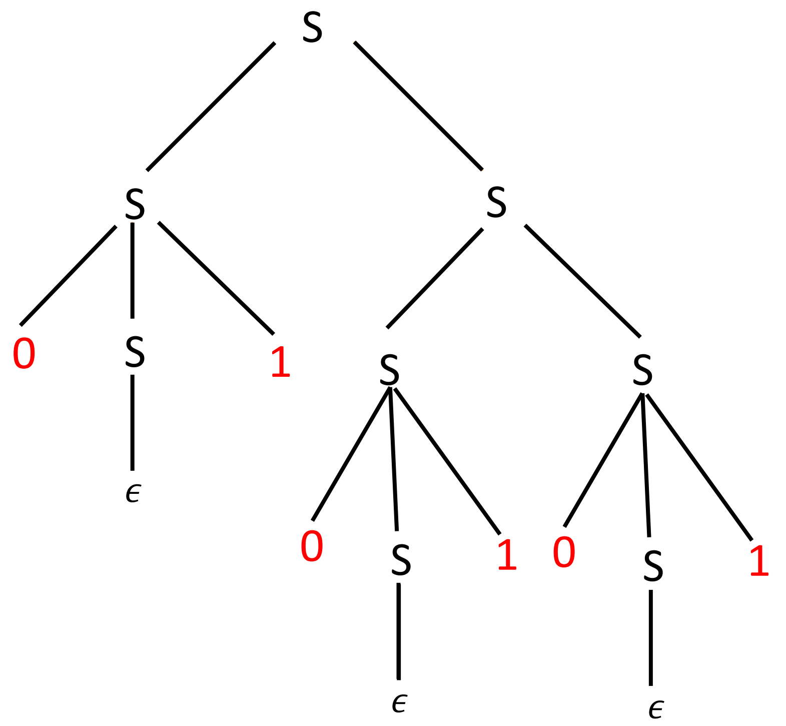
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

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

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: $S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{0S1}S \rightarrow 01\boldsymbol{S} \rightarrow 01\boldsymbol{SS} \rightarrow 01\boldsymbol{0S1}S \rightarrow 0101\boldsymbol{S} \rightarrow 0101\boldsymbol{0S1} \rightarrow \mathbf{010101}$

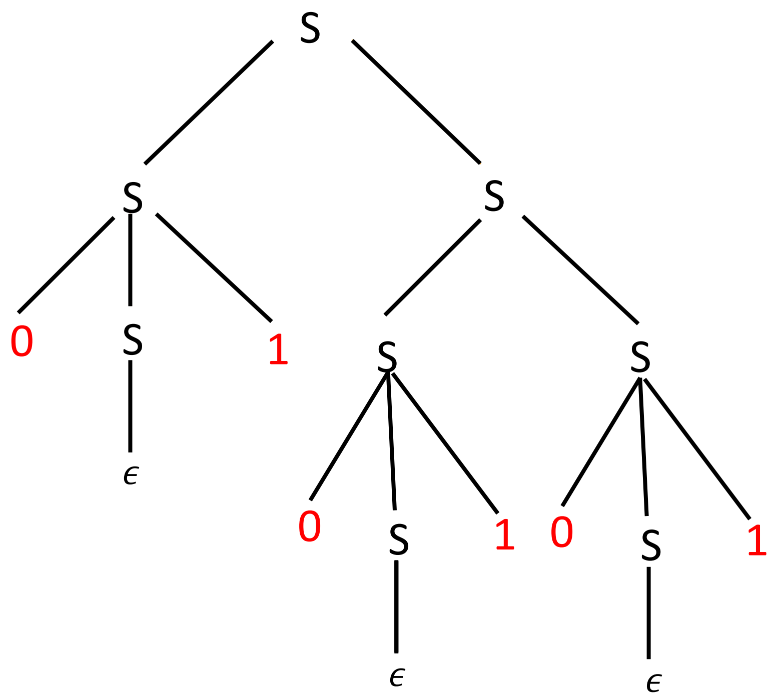
Parse trees for CFG

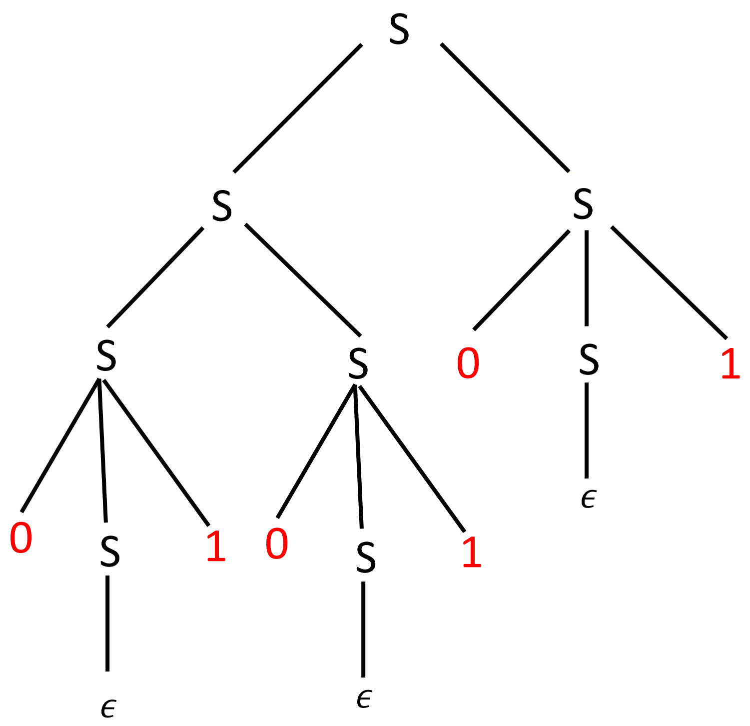
Ambiguous grammars: A CFG $\boldsymbol{G}$ is said to be **ambiguous** if there exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$**,** such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two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

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

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.**



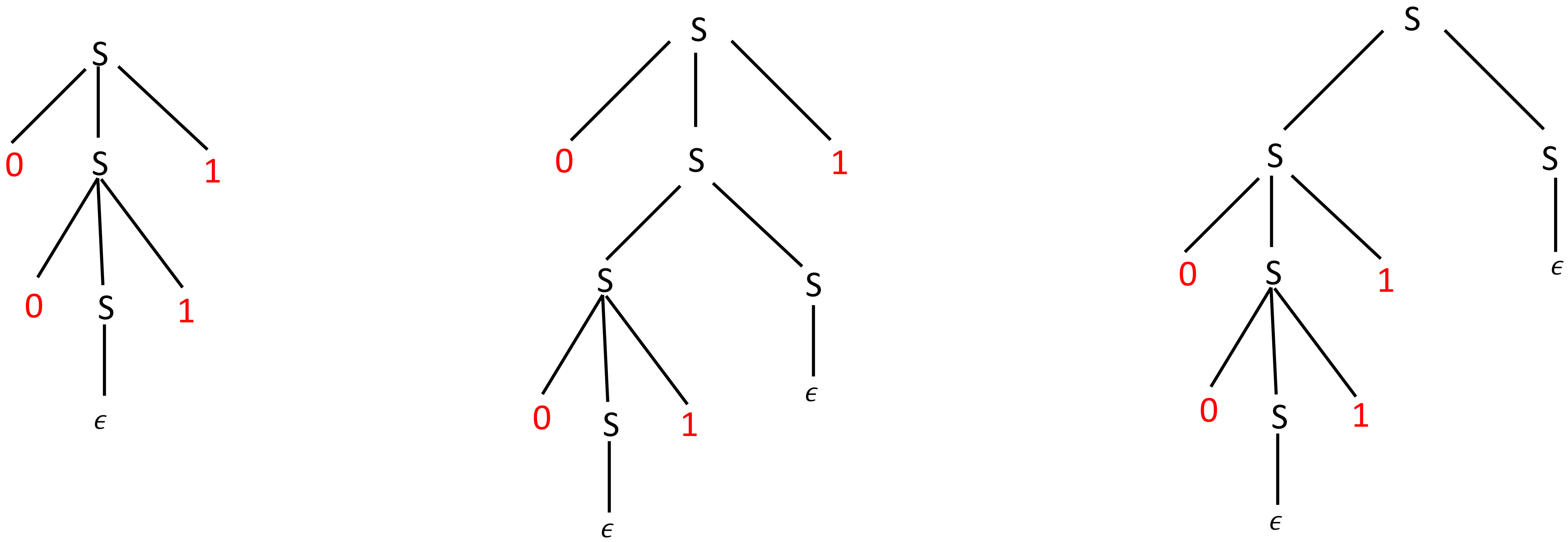


Leftmost Derivation: $S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{SSS} \rightarrow \boldsymbol{0S1}SS \rightarrow 01SS \rightarrow 01\boldsymbol{0S1}S \rightarrow 0101\boldsymbol{S} \rightarrow 0101\boldsymbol{0S1} \rightarrow \mathbf{010101}$

Parse trees for CFG

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

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



LD: $S \rightarrow \boldsymbol{0S1} \rightarrow 0\boldsymbol{0S1}1 \rightarrow \boldsymbol{0011}$

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

LD: $S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{0S1}S \rightarrow 0\boldsymbol{0S1}1S \rightarrow 0011\boldsymbol{S} \rightarrow \boldsymbol{0011}$

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

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

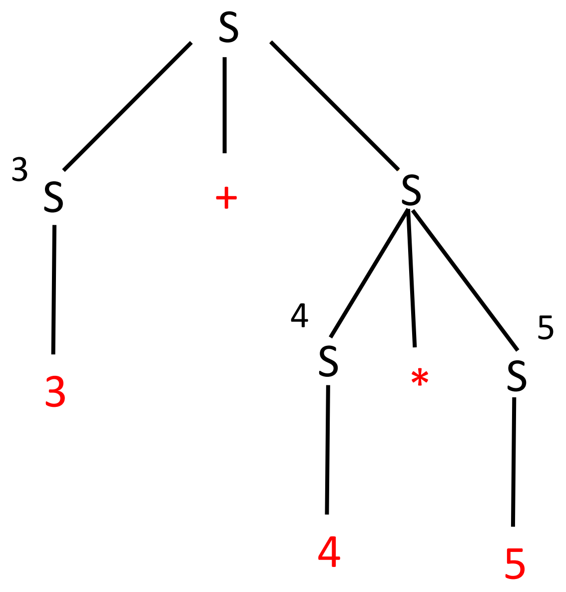
- What will be the result obtained from each of these ***parsings***?

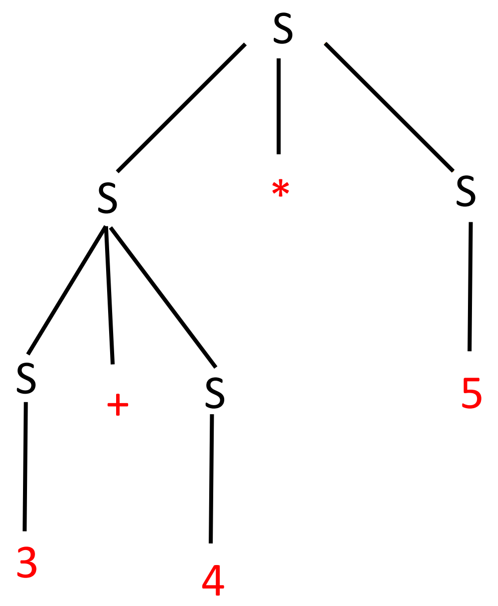
Ambiguity

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

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

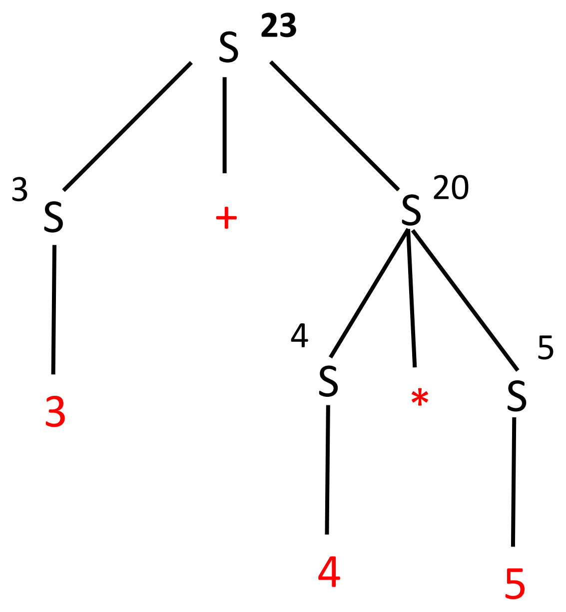
- If the compiler compiles the left parse tree

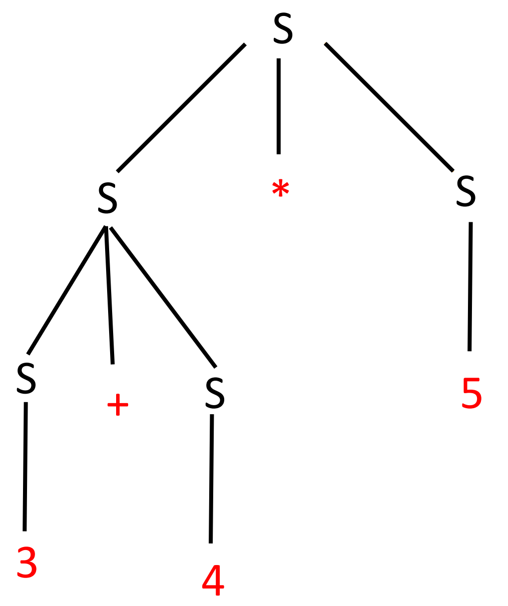
Ambiguity

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 $\mathbf{3 + 4 * 5}$





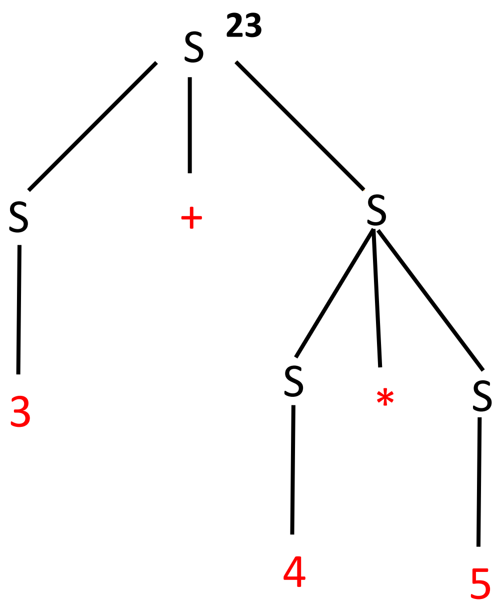
- If the compiler compiles the left parse tree. Outcome $= \mathbf{23}$

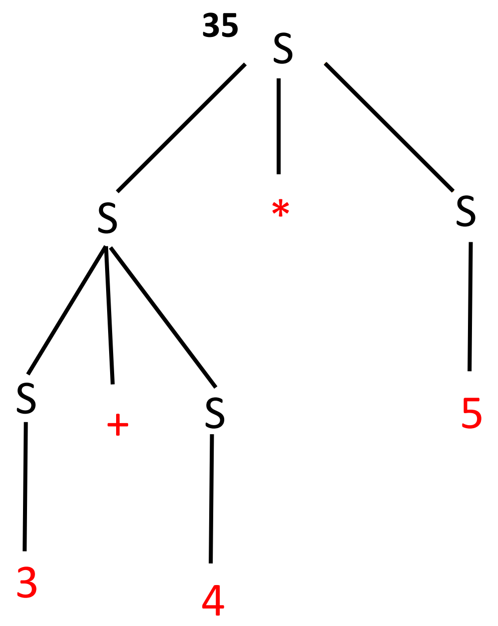
Ambiguity

Ambiguity may not be desirable.

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

and the derivation of the string $3 + 4 * 5$





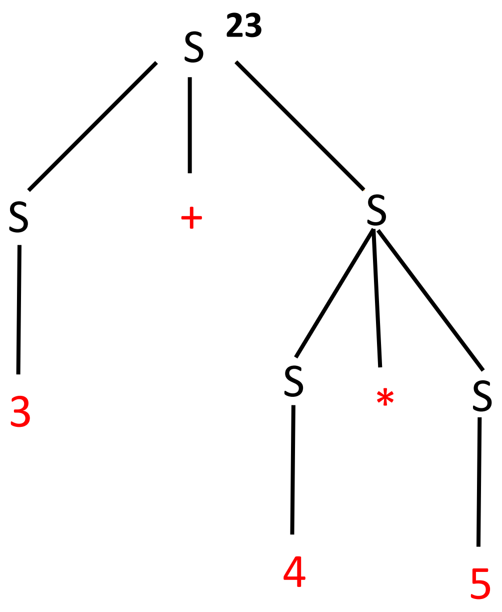
- If the compiler compiles the **right** parse tree. Outcome $= 35$

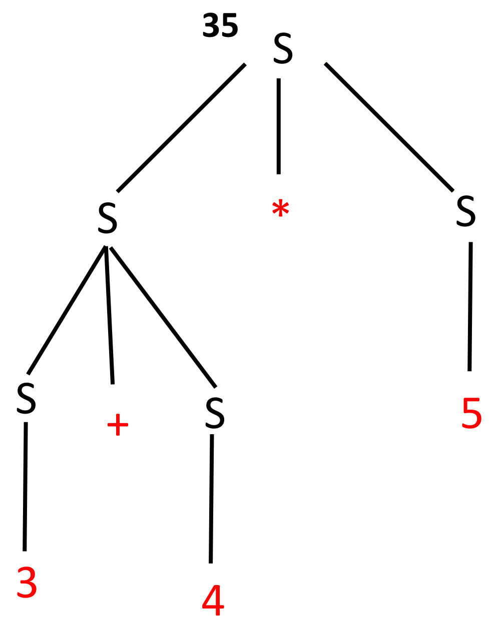
Ambiguity

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 $\boldsymbol{3 + 4 * 5}$





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

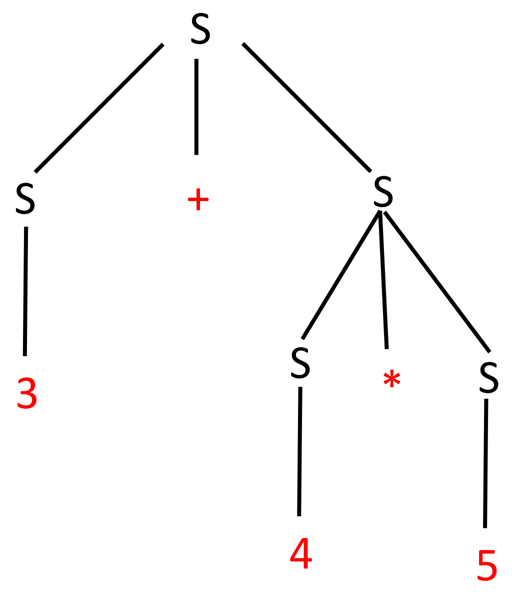
Ambiguity

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)

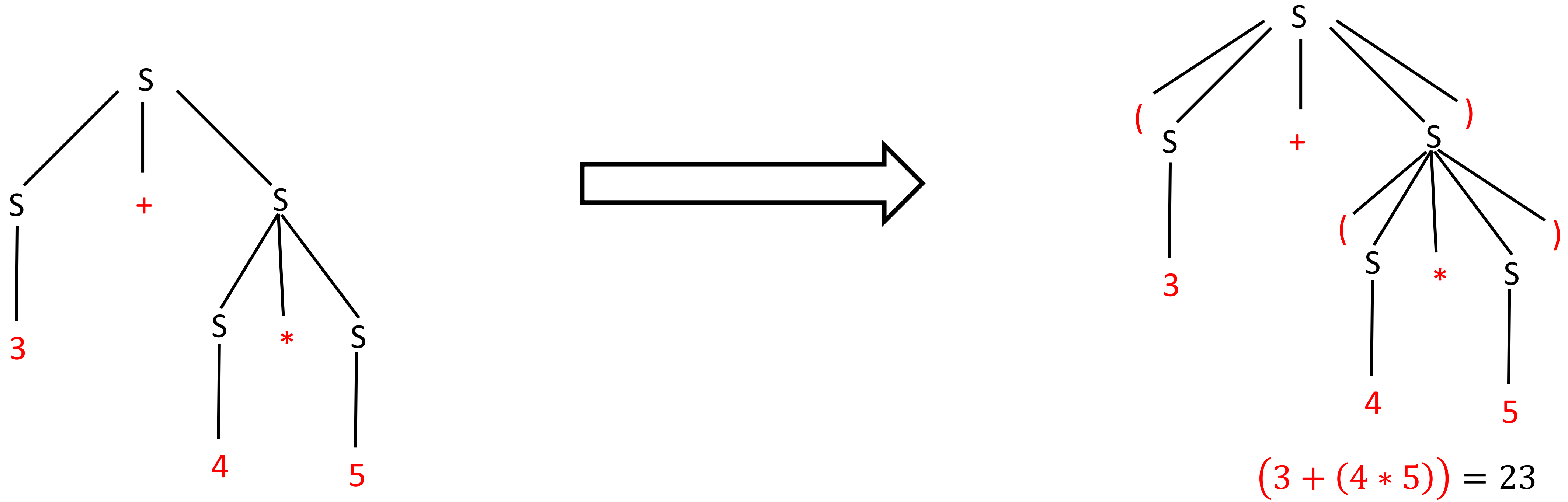
Ambiguity

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

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

1) Add parenthesis

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



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

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

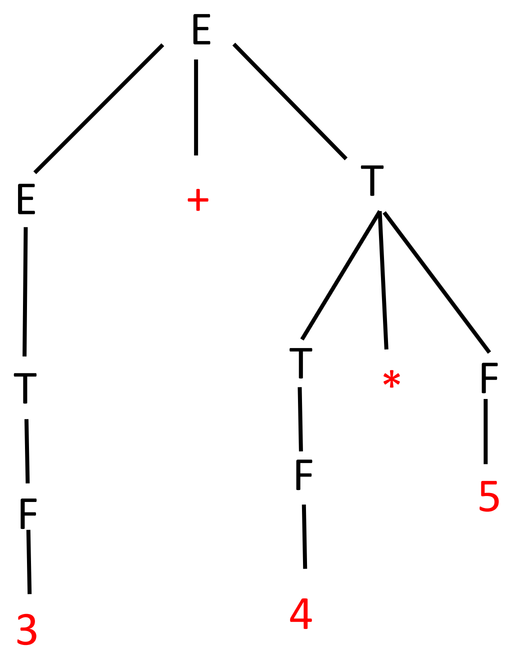
How can we get rid of this ambiguity? Change the production rules

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: $\boldsymbol{3 + (4 * 5)}$



Ambiguity

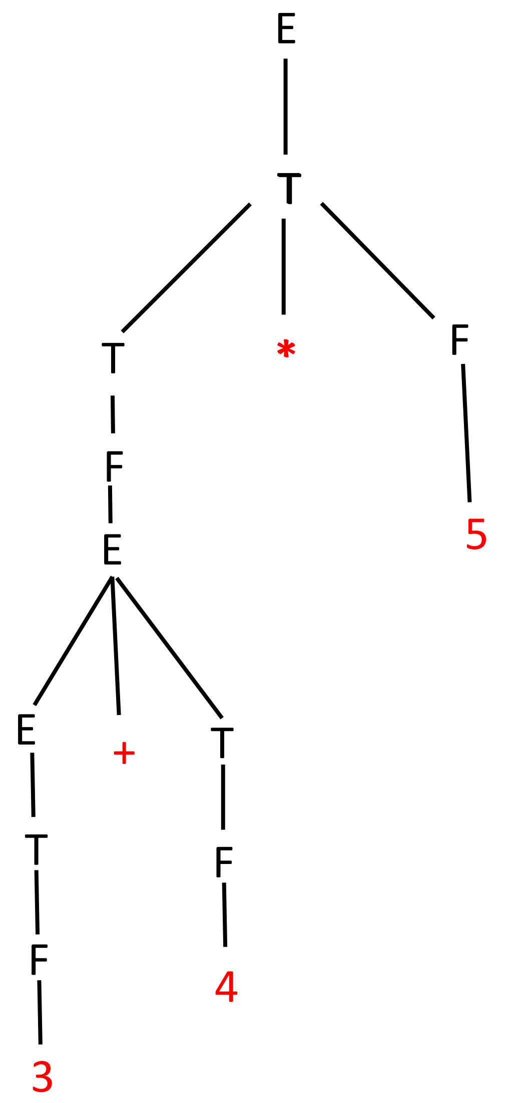
How can we get rid of this ambiguity? Change the production rules

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

New Grammar:

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

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



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)**
- A CFL L' is **inherently ambiguous** if all grammars G such that $L(G) = L'$ are ambiguous.
- 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 $2n - 1$ steps for generating strings $w \in L(G)$ of length n.**

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

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

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 $\boldsymbol{2n-1}$ 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 $|w| = k + 1$.

Since $|w| > 1$, any derivation will start from the rule $A \to BC$. So $\boldsymbol{w = xy}$**, where** $\boldsymbol{B \overset{*}{\Rightarrow} x, |x| > 0}$ **and** $\boldsymbol{C \overset{*}{\Rightarrow} y}, |\boldsymbol{y}| > \boldsymbol{0}$. 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

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

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

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 $S' \rightarrow S$

2. **Remove ϵ rules of the form $A \rightarrow \epsilon$**

We remove the rule $A \rightarrow \epsilon$. For each occurrence of A in the right side of the rule, we add a new rule with the occurrence of A deleted.

E.g.: Consider any rule $B \rightarrow uAvAw$ (u, v, w can be strings of variables and terminals)

Then new rules: $B \rightarrow uAvAw|uvAw|uAvw|uvw$

What if you had a rule such as $B \rightarrow A$? Then we would have needed to add a rule $B \rightarrow \epsilon$ (unless this rule has been already removed) as B is a **nullable variable.**

Repeat this procedure, until all ϵ-rules are removed.

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 $A \to \epsilon$**

E.g.: $S \to 0|X0|ZYZ$
$X \to Y|\epsilon$
$Y \to 1|X$

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:

Remove ϵ rules of the form $A \rightarrow \epsilon$ (For each occurrence of A in the right side of the rule, add a new rule with the occurrence of A deleted ; Remove nullable variables, Repeat the procedure until all ϵ rules are removed)

E.g.:

$$S \rightarrow 0|X0|ZYZ$$
$$X \rightarrow Y|\epsilon$$
$$Y \rightarrow 1|X$$

To remove $X \rightarrow \epsilon$, we add new rules:

$$S \rightarrow 0|X0|ZYZ$$
$$X \rightarrow Y$$
$$Y \rightarrow 1|X|\epsilon$$

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:

Remove ϵ rules of the form $A \to \epsilon$ (For each occurrence of A in the right side of the rule, add a new rule with the occurrence of A deleted ; Remove nullable variables, Repeat the procedure until all ϵ rules are removed)

E.g.:

$$S \to 0|X0|ZYZ$$
$$X \to Y|\epsilon$$
$$Y \to 1|X$$

To remove $X \to \epsilon$, we add new rules:

$$S \to 0|X0|ZYZ$$
$$X \to Y$$
$$Y \to 1|X|\epsilon$$

To remove $Y \to \epsilon$, we add:

$$S \to 0|X0|ZYZ|ZZ$$
$$X \to Y$$
$$Y \to 1|X$$

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 $S' \rightarrow S$

2. Remove ϵ rules of the form $A \rightarrow \epsilon$

3. **Remove unit rules of the form $\boldsymbol{A \rightarrow B}$**

We **remove the rule $\boldsymbol{A \rightarrow B}$** and **whenever a rule $\boldsymbol{B \rightarrow u}$ appears** (u is a string of terminals and variables), we **add a new rule $\boldsymbol{A \rightarrow u}$**, unless this rule was already removed.

Repeat these steps until all unit rules are removed.

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:

Remove unit rules of the form $\boldsymbol{A} \rightarrow \boldsymbol{B}$ (Whenever a rule $B \rightarrow u$ appears, we add a new rule $A \rightarrow u$, unless this rule was already removed. Repeat these steps until all unit rules are removed.)

E.g.:

$$S \rightarrow A|11$$
$$A \rightarrow B|1$$
$$B \rightarrow S|0$$

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:

Remove unit rules of the form $\boldsymbol{A \rightarrow B}$ (Whenever a rule $B \rightarrow u$ appears, we add a new rule $A \rightarrow u$, unless this rule was already removed. Repeat these steps until all unit rules are removed.)

E.g.:

$$\begin{gathered} S \rightarrow A|11 \\ A \rightarrow B|1 \\ B \rightarrow S|0 \end{gathered}$$

Remove $\boldsymbol{S \rightarrow A}$	Remove $\boldsymbol{A \rightarrow B}$	Remove $\boldsymbol{B \rightarrow S}$	Remove $\boldsymbol{B \rightarrow B}$	Remove $\boldsymbol{S \rightarrow B}$	Remove $\boldsymbol{A \rightarrow S}$
$S \rightarrow 11\boldsymbol{\mid B \mid 1}$	$S \rightarrow 11\mid B\mid 1$	$S \rightarrow 11\mid B\mid 1$	$S \rightarrow 11\mid B\mid 1$	$S \rightarrow 11\boldsymbol{\mid 0\mid}1$	$S \rightarrow 11\boldsymbol{\mid 0\mid}1$
$A \rightarrow B\mid 1$	$A \rightarrow 1\boldsymbol{\mid S\mid 0}$	$A \rightarrow 1\mid S\mid 0$	$A \rightarrow 1\mid S\mid 0$	$A \rightarrow 1\mid S\mid 0$	$A \rightarrow 1\boldsymbol{\mid 11\mid}0$
$B \rightarrow S\mid 0$	$B \rightarrow S\mid 0$	$B \rightarrow 0\boldsymbol{\mid 11\mid 1\mid B}$	$B \rightarrow 0\mid 11\mid 1$	$B \rightarrow 0\mid 11\mid 1$	$B \rightarrow 0\mid 11\mid 1$

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:

Remove unit rules of the form $\boldsymbol{A \to B}$ (Whenever a rule $B \to u$ appears, we add a new rule $A \to u$, unless this rule was already removed. Repeat these steps until all unit rules are removed.)

$$\begin{array}{l} S \to A|11 \\ A \to B|1 \\ B \to S|0 \end{array} \quad\Longrightarrow\quad \begin{array}{l} \boldsymbol{S \to 11|0|1} \\ \boldsymbol{A \to 1|11|0} \\ \boldsymbol{B \to 0|11|1} \end{array}$$

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 $A \to \epsilon$

3. Remove unit rules of the form $A \to B$

4. **Remove long rules of the form $\boldsymbol{A \to u_1 u_2 \cdots u_k}$**

Note that each u_i could be a variable or a terminal. We do the following:

- Replace $A \to u_1 u_2 \cdots u_k$, $(k \geq 3)$ with the rules $A \to u_1 A_1,\ A_1 \to u_2 A_2, \cdots, A_{k-2} \to u_{k-1} u_k$

- We replace any terminal u_i in the preceding rules with the new variable U_i and add the rule $U_i \to u_i$

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:

Add a new start variable $\boldsymbol{S' \rightarrow S}$

Remove $\boldsymbol{\epsilon}$ rules of the form $\boldsymbol{A \rightarrow \epsilon}$ (For each occurrence of A in the right side of the rule, add a new rule with the occurrence of A deleted ; Remove nullable variables, Repeat the procedure until all ϵ rules are removed)**.**

Remove unit rules of the form $\boldsymbol{A \rightarrow B}$ (Whenever a rule $B \rightarrow u$ appears, we add a new rule $A \rightarrow u$, unless this rule was already removed. Repeat these steps until all unit rules are removed.)

Remove long rules of the form $\boldsymbol{A \rightarrow u_1 u_2 \cdots u_k}$ (Replace $A \rightarrow u_1 u_2 \cdots u_k$, $(k \geq 3)$ with the rules $A \rightarrow u_1 A_1$, $A_1 \rightarrow u_2 A_2, \cdots, A_{k-2} \rightarrow u_{k-1} u_k$; Replace any terminal u_i in the preceding rules with the new variable U_i and add the rule $U_i \rightarrow u_i$).

Chomsky Normal Form

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

Convert the CFG

$$\boldsymbol{S \to ASA|aB}$$
$$\boldsymbol{A \to B|S}$$
$$\boldsymbol{B \to b|\epsilon}$$

to CNF.

1. Add a new start variable

$$\boldsymbol{S' \to S}$$
$$S \to ASA|aB$$
$$A \to B|S$$
$$B \to b|\epsilon$$

2a. Remove ϵ rules ($B \to \epsilon$)

$$S' \to S$$
$$S \to ASA|aB|\boldsymbol{a}$$
$$A \to B|S|\boldsymbol{\epsilon}$$
$$B \to b$$

2b. Remove ϵ rules (A$\to \epsilon$)

$$S' \to S$$
$$S \to ASA|aB|a|\boldsymbol{AS}|\boldsymbol{SA}|\boldsymbol{S}$$
$$A \to B|S$$
$$B \to b$$

Chomsky Normal Form

CNF:

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

Convert the CFG

$$\boldsymbol{S \rightarrow ASA|aB}$$
$$\boldsymbol{A \rightarrow B|S}$$
$$\boldsymbol{B \rightarrow b|\epsilon}$$

to CNF.

3a. Remove $\boldsymbol{S \rightarrow S}$

$$S' \rightarrow S$$
$$S \rightarrow ASA|aB|a|AS|SA$$
$$A \rightarrow B|S$$
$$B \rightarrow b$$

3b. Remove $\boldsymbol{S' \rightarrow S}$

$$S' \rightarrow \boldsymbol{ASA|aB|a|AS|SA}$$
$$S \rightarrow ASA|aB|a|AS|SA$$
$$A \rightarrow B|S$$
$$B \rightarrow b$$

3c. Remove A→ $\boldsymbol{B}$

$$S' \rightarrow ASA|aB|a|AS|SA$$
$$S \rightarrow ASA|aB|a|AS|SA$$
$$A \rightarrow S|\boldsymbol{b}$$
$$B \rightarrow b$$

3d. Remove A→ $\boldsymbol{S}$

$$S' \rightarrow ASA|aB|a|AS|SA$$
$$S \rightarrow ASA|aB|a|AS|SA$$
$$A \rightarrow b|\boldsymbol{ASA|aB|a|AS|SA}$$
$$B \rightarrow b$$

Chomsky Normal Form

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

Convert the CFG

$$\boldsymbol{S \to ASA|aB}$$
$$\boldsymbol{A \to B|S}$$
$$\boldsymbol{B \to b|\epsilon}$$

to CNF.

3d. Remove A→ $\boldsymbol{S}$

$$S' \to ASA|aB|a|AS|SA$$
$$S \to ASA|aB|a|AS|SA$$
$$A \to b|ASA|aB|a|AS|SA$$
$$B \to b$$

4a. Remove long rules

$$S' \to A\boldsymbol{SA}|aB|a|AS|SA$$
$$S \to ASA|aB|a|AS|SA$$
$$A \to b|ASA|aB|a|AS|SA$$
$$B \to b$$

There are other rules of the form: Var → ASA

4b. Remove long rules

$$S' \to A\boldsymbol{U}|aB|a|AS|SA$$
$$S \to A\boldsymbol{U}|aB|a|AS|SA$$
$$A \to b|A\boldsymbol{U}|aB|a|AS|SA$$
$$U \to SA$$
$$B \to b$$

4c. Remove long rules

$$S' \to AU|VB|a|AS|SA$$
$$S \to A\boldsymbol{U}|VB|a|AS|SA$$
$$A \to b|A\boldsymbol{U}|VB|a|AS|SA$$
$$U \to SA$$
$$\boldsymbol{V \to a}$$
$$B \to b$$

Chomsky Normal Form

CNF: $\boldsymbol{A \to BC}$ [B, C are not start variables]

$\boldsymbol{A \to a}$ [a is a terminal]

$\boldsymbol{S \to \epsilon}$ [S is the Start Variable]

Convert the CFG

$$\boldsymbol{S \to ASA|aB}$$
$$\boldsymbol{A \to B|S}$$
$$\boldsymbol{B \to b|\epsilon}$$

to CNF.

$$\boldsymbol{S' \to AU|VB|a|AS|SA}$$
$$\boldsymbol{S \to AU|VB|a|AS|SA}$$
$$\boldsymbol{A \to b|AU|VB|a|AS|SA}$$
$$\boldsymbol{U \to SA}$$
$$\boldsymbol{V \to a}$$
$$\boldsymbol{B \to b}$$

Pushdown Automata

- For regular languages we had
  - Designed Finite automata (DFA, NFA) that recognize the strings by the language. Helped us decide whether a given string ω belongs to the language.

Pushdown Automata

- For regular languages we had
  - Designed Finite automata (DFA, NFA) that recognize the strings by the language. Helped us decide whether a given string ω belongs to the language.
  - Developed regular expressions/linear grammar that can generate all the strings in the language.

Pushdown Automata

- For regular languages we had
  - Designed Finite automata (DFA, NFA) that recognize the strings by the language. Helped us decide whether a given string ω belongs to the language.
  - Developed regular expressions/linear grammar that can generate all the strings in the language.
- For context free languages,
  - Context Free Grammars generate all the strings in the language

Pushdown Automata

- For regular languages we had
  - Designed Finite automata (DFA, NFA) that recognize the strings by the language. Helped us decide whether a given string ω belongs to the language.
  - Developed regular expressions/linear grammar that can generate all the strings in the language.
- For context free languages,
  - Context Free Grammars generate all the strings in the language
  - Can we build an automata that recognizes **exactly** context free languages?

Pushdown Automata

- For regular languages we had
  - Designed Finite automata (DFA, NFA) that recognize the strings by the language. Helped us decide whether a given string ω belongs to the language.
  - Developed regular expressions/linear grammar that can generate all the strings in the language.
- For context free languages,
  - Context Free Grammars generate all the strings in the language
  - Can we build an automata that recognizes **exactly** context free languages?

- **Finite Automaton model recognizes ALL regular languages**
- Any automata that recognizes **ALL** context free languages will need unbounded memory.

Pushdown Automata

- Finite Automaton model recognizes ALL regular languages
- Any automata that recognizes **ALL** context free languages will need unbounded memory.

Intuition to build an Automata for CFL

- It should be some **Finite State Machine** that has access to a memory device with infinite memory, i.e.

Automata for CFL = FSM + Memory device

- **FSM may choose to ignore the memory device** completely in which case it behaves like a DFA/NFA.
- FSM makes use of the Memory device to recognize “non-Regular” CFLs.

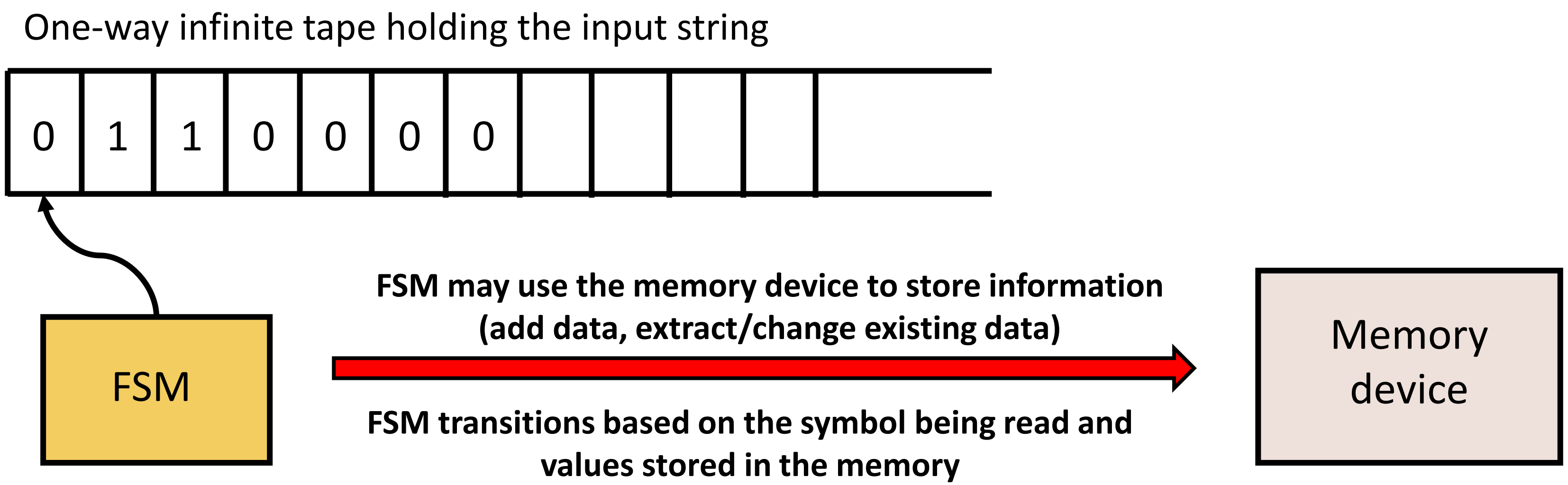
E.g.: $\{0^n1^n, n \in \mathbb{N}\}$

Pushdown Automata

Intuition to build an Automata for CFL

- **Automata for CFL = FSM + Memory device**
- **FSM may choose to ignore the memory device** completely in which case it behaves like a DFA/NFA.
- FSM makes use of the Memory device to recognize “non-Regular” CFLs.

E.g.: $\{0^n 1^n, n \in \mathbb{N}\}$



Thank You!