CS 383

Lecture 09 – Context-free grammars

Stephen Checkoway

Spring 2024

Context-free grammars (CFGs)

Method of generating (or describing) languages by giving rules to derive strings

Rules contain

- **Terminals** symbols from an alphabet (written in `typewriter` font)
- **Variables** which expand to sequences of terminals and variables (typically upper case letters)

Rules have a variable on the left, an arrow ($\to$), and a sequence of terminals and variables on the right

Example:

Context-free grammars (CFGs)

Method of generating (or describing) languages by giving rules to derive strings

Rules contain

Terminals symbols from an alphabet (written in `typewriter` font)

Variables which expand to sequences of terminals and variables (typically upper case letters)

Rules have a variable on the left, an arrow ($\to$), and a sequence of terminals and variables on the right

Example:

$$S \to AB$$
$$A \to \texttt{a}A$$
$$A \to \varepsilon$$
$$B \to \texttt{b}B$$
$$B \to \varepsilon$$

Context-free grammars (CFGs)

Method of generating (or describing) languages by giving rules to derive strings

Rules contain

- **Terminals** symbols from an alphabet (written in `typewriter` font)
- **Variables** which expand to sequences of terminals and variables (typically upper case letters)

Rules have a variable on the left, an arrow (→), and a sequence of terminals and variables on the right

Example:

$$S \to AB$$
$$A \to \texttt{a}A$$
$$A \to \varepsilon$$
$$B \to \texttt{b}B$$
$$B \to \varepsilon$$

We often combine multiple rules with the same left-hand side using |

$$S \to AB$$
$$A \to \texttt{a}A \mid \varepsilon$$
$$B \to \texttt{b}B \mid \varepsilon$$

Deriving strings

A CFG derives a string by starting with the start variable (usually the variable on the left in the first rule) and applying rules until no variables remain

The CFG

$$
\begin{aligned}
S &\to AB \\
A &\to \mathtt{a}A \mid \varepsilon \\
B &\to \mathtt{b}B \mid \varepsilon
\end{aligned}
$$

derives the following strings

$$
\begin{aligned}
&S \Rightarrow AB \Rightarrow \varepsilon B \Rightarrow \varepsilon\varepsilon = \varepsilon \\
&S \Rightarrow AB \Rightarrow \mathtt{a}AB \Rightarrow \mathtt{a}\varepsilon B \Rightarrow \mathtt{a}\varepsilon\varepsilon = \mathtt{a} \\
&S \Rightarrow AB \Rightarrow \mathtt{a}AB \Rightarrow \mathtt{aa}AB \Rightarrow \mathtt{aa}\varepsilon B \Rightarrow \mathtt{aab}B \Rightarrow \mathtt{aab}\varepsilon = \mathtt{aab} \\
&\quad\vdots
\end{aligned}
$$

Derivations

The order in which we replace a variable in a derivation with the RHS of a production rule doesn't matter[1]

In a left-most derivation, we replace the left-most variable in each step

In a right-most derivation, we replace the right-most variable in each step

[1] except in one case we'll get to

Left-most/right-most derivation example

$$S \to ST \mid \mathtt{a}T\mathtt{a}$$
$$T \to S \mid \mathtt{a}T\mathtt{a} \mid \mathtt{b}$$

Left-most derivation of `aabaaaba`:

$$S \Rightarrow$$

Left-most/right-most derivation example

$$S \to ST \mid \mathtt{a}T\mathtt{a}$$
$$T \to S \mid \mathtt{a}T\mathtt{a} \mid \mathtt{b}$$

Left-most derivation of aabaaaba:

$$S \Rightarrow ST$$

Left-most/right-most derivation example

$$S \to ST \mid \mathtt{a}T\mathtt{a}$$
$$T \to S \mid \mathtt{a}T\mathtt{a} \mid \mathtt{b}$$

Left-most derivation of `aabaaaba`:

$$S \Rightarrow ST$$
$$\Rightarrow \mathtt{a}T\mathtt{a}T$$

Left-most/right-most derivation example

$$S \to ST \mid \texttt{a}T\texttt{a}$$
$$T \to S \mid \texttt{a}T\texttt{a} \mid \texttt{b}$$

Left-most derivation of `aabaaaba`:

$$\begin{aligned} S &\Rightarrow ST \\ &\Rightarrow \texttt{a}T\texttt{a}T \\ &\Rightarrow \texttt{aa}T\texttt{aa}T \end{aligned}$$

Left-most/right-most derivation example

$$S \to ST \mid \mathtt{a}T\mathtt{a}$$
$$T \to S \mid \mathtt{a}T\mathtt{a} \mid \mathtt{b}$$

Left-most derivation of `aabaaaba`:

$$\begin{aligned} S &\Rightarrow ST \\ &\Rightarrow \mathtt{a}T\mathtt{a}T \\ &\Rightarrow \mathtt{aa}T\mathtt{aa}T \\ &\Rightarrow \mathtt{aabaa}T \end{aligned}$$

Left-most/right-most derivation example

$$S \to ST \mid \mathtt{a}T\mathtt{a}$$
$$T \to S \mid \mathtt{a}T\mathtt{a} \mid \mathtt{b}$$

Left-most derivation of aabaaaba:

$$\begin{aligned} S &\Rightarrow ST \\ &\Rightarrow \mathtt{a}T\mathtt{a}T \\ &\Rightarrow \mathtt{aa}T\mathtt{aa}T \\ &\Rightarrow \mathtt{aabaa}T \\ &\Rightarrow \mathtt{aabaaa}T\mathtt{a} \end{aligned}$$

Left-most/right-most derivation example

$$S \to ST \mid \mathtt{a}T\mathtt{a}$$
$$T \to S \mid \mathtt{a}T\mathtt{a} \mid \mathtt{b}$$

Left-most derivation of `aabaaaba`:

$$\begin{aligned} S &\Rightarrow ST \\ &\Rightarrow \mathtt{a}T\mathtt{a}T \\ &\Rightarrow \mathtt{aa}T\mathtt{aa}T \\ &\Rightarrow \mathtt{aabaa}T \\ &\Rightarrow \mathtt{aabaaa}T\mathtt{a} \\ &\Rightarrow \mathtt{aabaaaba} \end{aligned}$$

Left-most/right-most derivation example

$$S \to ST \mid \mathtt{a}T\mathtt{a}$$
$$T \to S \mid \mathtt{a}T\mathtt{a} \mid \mathtt{b}$$

Left-most derivation of aabaaaba:

$$\begin{aligned} S &\Rightarrow ST \\ &\Rightarrow \mathtt{a}T\mathtt{a}T \\ &\Rightarrow \mathtt{aa}T\mathtt{aa}T \\ &\Rightarrow \mathtt{aabaa}T \\ &\Rightarrow \mathtt{aabaaa}T\mathtt{a} \\ &\Rightarrow \mathtt{aabaaaba} \end{aligned}$$

Right-most derivation of aabaaaba:

$$\begin{aligned} S &\Rightarrow ST \\ &\Rightarrow S\mathtt{a}T\mathtt{a} \\ &\Rightarrow S\mathtt{aba} \\ &\Rightarrow \mathtt{a}T\mathtt{aaba} \\ &\Rightarrow \mathtt{aa}T\mathtt{aaaba} \\ &\Rightarrow \mathtt{aabaaaba} \end{aligned}$$

Another example

The CFG

$$S \to \mathtt{a}S\mathtt{b} \mid \varepsilon$$

derives

$$S \Rightarrow \varepsilon$$
$$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{ab}$$
$$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow \mathtt{aabb}$$
$$\vdots$$
$$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \cdots \Rightarrow \mathtt{a}^n S \mathtt{b}^n \Rightarrow \mathtt{a}^n\mathtt{b}^n$$
$$\vdots$$

The language of this CFG is $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$

Nested brackets

Given the alphabet $\Sigma = \{(,), [,]\}$, design a CFG that generates the language of properly nested brackets.

- ε
- ()
- []
- ([])[](())
- ...

Nested brackets

Given the alphabet $\Sigma = \{(,), [,]\}$, design a CFG that generates the language of properly nested brackets.

- ε
- ()
- []
- ([])[](())
- ...

$$S \to P \mid B \mid SS \mid \varepsilon$$
$$P \to (S)$$
$$B \to [S]$$

More CFG examples

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ Construct a CFG for the languages over Σ

- $A = \Sigma^*$
- $B = \{w \mid w$ contains at least three $\texttt{b}$s$\}$
- $C = \{w \mid w$ starts and ends with different symbols$\}$
- $D = \{w \mid$ the length of w is odd and the middle symbol is $\texttt{b}\}$
- $E = \{w \mid w = w^{\mathcal{R}}\}$
- $F = \varnothing$

Formally speaking

A CFG is a 4-tuple $G = (V, \Sigma, R, S)$ where

- V is a finite set of variables (or nonterminals)
- Σ is a finite set of terminals ($V \cap \Sigma = \varnothing$)
- R is a finite set of production rules
- $S \in V$ is the start variable

Formally speaking

A CFG is a 4-tuple $G = (V, \Sigma, R, S)$ where

- V is a finite set of variables (or nonterminals)
- Σ is a finite set of terminals ($V \cap \Sigma = \varnothing$)
- R is a finite set of production rules
- $S \in V$ is the start variable

If $u, v, w \in (\Sigma \cup V)^*$ and G has a rule $A \to v$, then we say uAw yields uvw and write $uAw \Rightarrow uvw$

Formally speaking

A CFG is a 4-tuple $G = (V, \Sigma, R, S)$ where

- V is a finite set of variables (or nonterminals)
- Σ is a finite set of terminals ($V \cap \Sigma = \varnothing$)
- R is a finite set of production rules
- $S \in V$ is the start variable

If $u, v, w \in (\Sigma \cup V)^*$ and G has a rule $A \to v$, then we say uAw yields uvw and write $uAw \Rightarrow uvw$

We say u derives v, written $u \overset{*}{\Rightarrow} v$ to mean either $u = v$ or there exist $u_1, u_2, \ldots, u_n \in (\Sigma \cup V)^*$ such that

$$u = u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_n = v$$

Formally speaking

A CFG is a 4-tuple $G = (V, \Sigma, R, S)$ where

- V is a finite set of variables (or nonterminals)
- Σ is a finite set of terminals ($V \cap \Sigma = \varnothing$)
- R is a finite set of production rules
- $S \in V$ is the start variable

If $u, v, w \in (\Sigma \cup V)^*$ and G has a rule $A \to v$, then we say uAw yields uvw and write $uAw \Rightarrow uvw$

We say u derives v, written $u \overset{*}{\Rightarrow} v$ to mean either $u = v$ or there exist $u_1, u_2, \ldots, u_n \in (\Sigma \cup V)^*$ such that

$$u = u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_n = v$$

The language of G is $L(G) = \{w \mid w \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w\}$
We say G generates a language A if $L(G) = A$

Arithmetic expressions

Given the alphabet $\Sigma = \{ (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$, design a CFG that generates the language of arithmetic expressions

- `37`
- `8+22-8/6`
- `10*(8-2)`
- ...

An expression can be a number or two expressions separated by an operator or a parenthesized expression
A number is one or more digits

Arithmetic expressions

Given the alphabet $\Sigma = \{\texttt{(}, \texttt{)}, \texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}\}$, design a CFG that generates the language of arithmetic expressions

- 37
- 8+22-8/6
- 10*(8-2)
- ...

An expression can be a number or two expressions separated by an operator or a parenthesized expression
A number is one or more digits

$$E \to N \mid E\texttt{*}E \mid E\texttt{/}E \mid E\texttt{+}E \mid E\texttt{-}E \mid \texttt{(}E\texttt{)}$$
$$N \to DN \mid D$$
$$D \to \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

Parse trees give a way to visualize a derivation

$$E \to N \mid E\texttt{*}E \mid E\texttt{/}E \mid E\texttt{+}E \mid E\texttt{-}E \mid \texttt{(}E\texttt{)}$$

$$N \to DN \mid D$$

$$D \to \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

Derivation

E

Parse tree

E

Parse trees give a way to visualize a derivation

$$E \rightarrow N \mid E*E \mid E/E \mid E+E \mid E-E \mid (E)$$

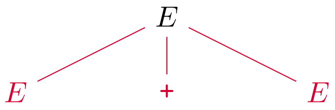
$$N \rightarrow DN \mid D$$

$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$E \Rightarrow E+E$$

Parse tree



Parse trees give a way to visualize a derivation

$$E \to N \mid E\texttt{*}E \mid E\texttt{/}E \mid E\texttt{+}E \mid E\texttt{-}E \mid \texttt{(}E\texttt{)}$$

$$N \to DN \mid D$$

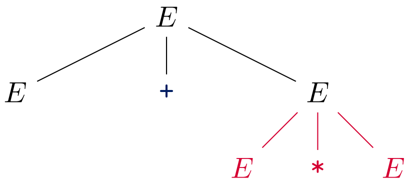
$$D \to \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

Derivation

$$E \Rightarrow E\texttt{+}E$$

$$\Rightarrow E\texttt{+}E\texttt{*}E$$

Parse tree



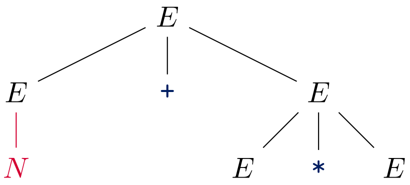
Parse trees give a way to visualize a derivation

$$E \rightarrow N \mid E\texttt{*}E \mid E\texttt{/}E \mid E\texttt{+}E \mid E\texttt{-}E \mid \texttt{(}E\texttt{)}$$
$$N \rightarrow DN \mid D$$
$$D \rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

Derivation

$$E \Rightarrow E\texttt{+}E$$
$$\Rightarrow E\texttt{+}E\texttt{*}E$$
$$\Rightarrow N\texttt{+}E\texttt{*}E$$

Parse tree



Parse trees give a way to visualize a derivation

$$E \to N \mid E{*}E \mid E/E \mid E{+}E \mid E{-}E \mid (E)$$

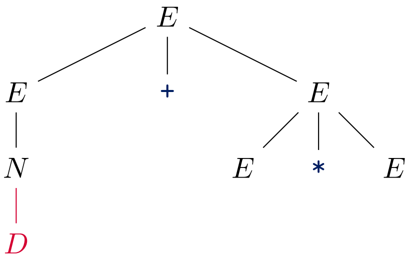
$$N \to DN \mid D$$

$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned} E &\Rightarrow E{+}E \\ &\Rightarrow E{+}E{*}E \\ &\Rightarrow N{+}E{*}E \\ &\Rightarrow D{+}E{*}E \end{aligned}$$

Parse tree



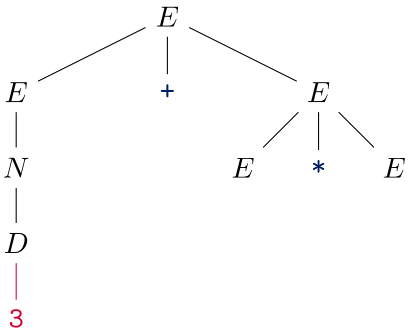
Parse trees give a way to visualize a derivation

$$E \to N \mid E{*}E \mid E/E \mid E{+}E \mid E{-}E \mid (E)$$
$$N \to DN \mid D$$
$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned} E &\Rightarrow E{+}E \\ &\Rightarrow E{+}E{*}E \\ &\Rightarrow N{+}E{*}E \\ &\Rightarrow D{+}E{*}E \\ &\Rightarrow 3{+}E{*}E \end{aligned}$$

Parse tree



Parse trees give a way to visualize a derivation

$$E \rightarrow N \mid E\texttt{*}E \mid E\texttt{/}E \mid E\texttt{+}E \mid E\texttt{-}E \mid \texttt{(}E\texttt{)}$$

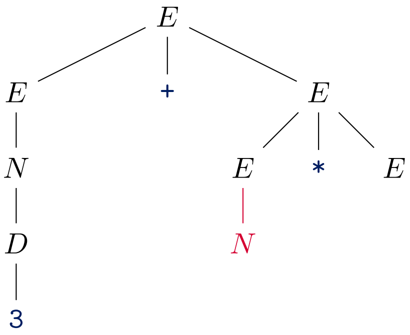
$$N \rightarrow DN \mid D$$

$$D \rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

Derivation

$$\begin{aligned} E &\Rightarrow E\texttt{+}E \\ &\Rightarrow E\texttt{+}E\texttt{*}E \\ &\Rightarrow N\texttt{+}E\texttt{*}E \\ &\Rightarrow D\texttt{+}E\texttt{*}E \\ &\Rightarrow \texttt{3+}E\texttt{*}E \\ &\Rightarrow \texttt{3+}N\texttt{*}E \end{aligned}$$

Parse tree



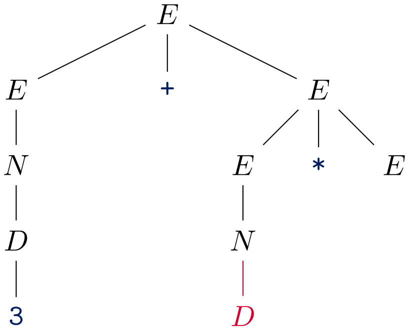
Parse trees give a way to visualize a derivation

$$E \to N \mid E{*}E \mid E/E \mid E{+}E \mid E{-}E \mid (E)$$
$$N \to DN \mid D$$
$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned} E &\Rightarrow E{+}E \\ &\Rightarrow E{+}E{*}E \\ &\Rightarrow N{+}E{*}E \\ &\Rightarrow D{+}E{*}E \\ &\Rightarrow 3{+}E{*}E \\ &\Rightarrow 3{+}N{*}E \\ &\Rightarrow 3{+}D{*}E \end{aligned}$$

Parse tree



Parse trees give a way to visualize a derivation

$$E \to N \mid E{*}E \mid E/E \mid E{+}E \mid E{-}E \mid (E)$$

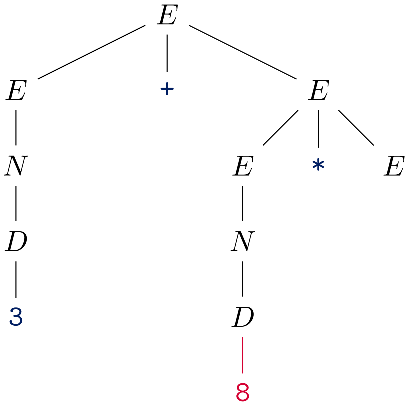
$$N \to DN \mid D$$

$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned} E &\Rightarrow E{+}E \\ &\Rightarrow E{+}E{*}E \\ &\Rightarrow N{+}E{*}E \\ &\Rightarrow D{+}E{*}E \\ &\Rightarrow 3{+}E{*}E \\ &\Rightarrow 3{+}N{*}E \\ &\Rightarrow 3{+}D{*}E \\ &\Rightarrow 3{+}8{*}E \end{aligned}$$

Parse tree



Parse trees give a way to visualize a derivation

$$E \to N \mid E{*}E \mid E/E \mid E{+}E \mid E{-}E \mid (E)$$

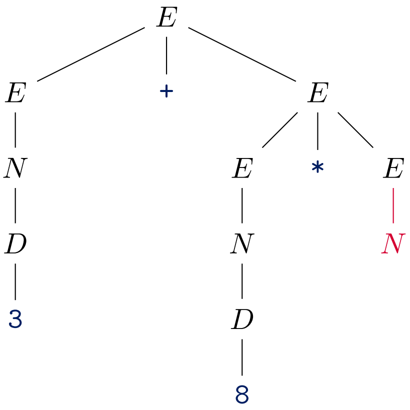
$$N \to DN \mid D$$

$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned}
E &\Rightarrow E{+}E \\
&\Rightarrow E{+}E{*}E \\
&\Rightarrow N{+}E{*}E \\
&\Rightarrow D{+}E{*}E \\
&\Rightarrow 3{+}E{*}E \\
&\Rightarrow 3{+}N{*}E \\
&\Rightarrow 3{+}D{*}E \\
&\Rightarrow 3{+}8{*}E \\
&\Rightarrow 3{+}8{*}N
\end{aligned}$$

Parse tree



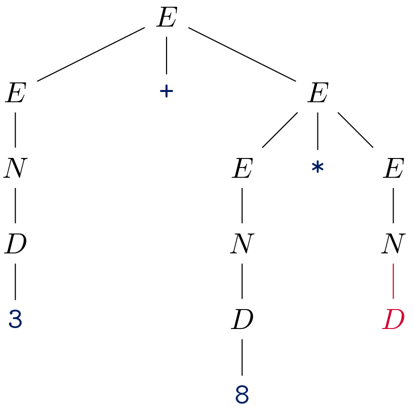
Parse trees give a way to visualize a derivation

$$E \to N \mid E*E \mid E/E \mid E+E \mid E-E \mid (E)$$
$$N \to DN \mid D$$
$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow E+E*E \\ &\Rightarrow N+E*E \\ &\Rightarrow D+E*E \\ &\Rightarrow 3+E*E \\ &\Rightarrow 3+N*E \\ &\Rightarrow 3+D*E \\ &\Rightarrow 3+8*E \\ &\Rightarrow 3+8*N \\ &\Rightarrow 3+8*D \end{aligned}$$

Parse tree



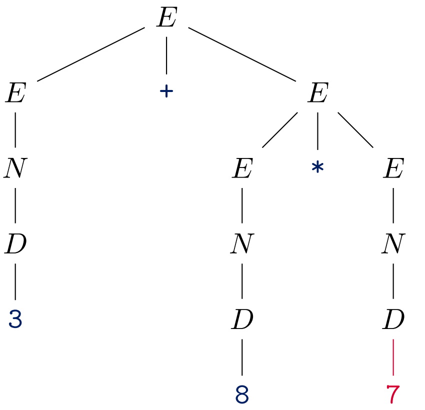
Parse trees give a way to visualize a derivation

$$E \to N \mid E{*}E \mid E/E \mid E{+}E \mid E{-}E \mid (E)$$
$$N \to DN \mid D$$
$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned}
E &\Rightarrow E{+}E \\
&\Rightarrow E{+}E{*}E \\
&\Rightarrow N{+}E{*}E \\
&\Rightarrow D{+}E{*}E \\
&\Rightarrow 3{+}E{*}E \\
&\Rightarrow 3{+}N{*}E \\
&\Rightarrow 3{+}D{*}E \\
&\Rightarrow 3{+}8{*}E \\
&\Rightarrow 3{+}8{*}N \\
&\Rightarrow 3{+}8{*}D \\
&\Rightarrow 3{+}8{*}7
\end{aligned}$$

Parse tree



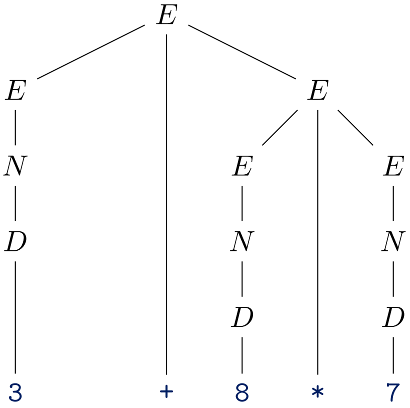
Parse trees give a way to visualize a derivation

$$E \to N \mid E{*}E \mid E/E \mid E{+}E \mid E{-}E \mid (E)$$
$$N \to DN \mid D$$
$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Derivation

$$\begin{aligned}
E &\Rightarrow E{+}E \\
&\Rightarrow E{+}E{*}E \\
&\Rightarrow N{+}E{*}E \\
&\Rightarrow D{+}E{*}E \\
&\Rightarrow 3{+}E{*}E \\
&\Rightarrow 3{+}N{*}E \\
&\Rightarrow 3{+}D{*}E \\
&\Rightarrow 3{+}8{*}E \\
&\Rightarrow 3{+}8{*}N \\
&\Rightarrow 3{+}8{*}D \\
&\Rightarrow 3{+}8{*}7
\end{aligned}$$

Parse tree



Different derivations can give rise to the same parse tree

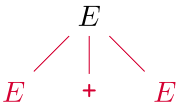
Two different derivations give the same parse tree

E E E E

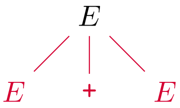
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$E \Rightarrow E\text{+}E$



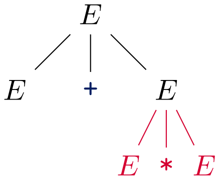
$E \Rightarrow E\text{+}E$



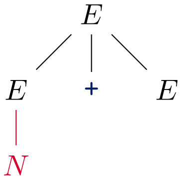
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$E \Rightarrow E{+}E$$
$$\Rightarrow E{+}E{*}E$$



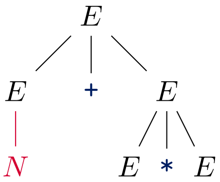
$$E \Rightarrow E{+}E$$
$$\Rightarrow N{+}E$$



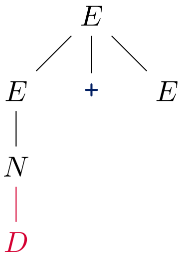
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow E+E*E \\
&\Rightarrow N+E*E
\end{aligned}
$$



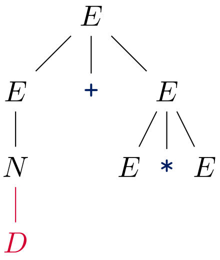
$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow N+E \\
&\Rightarrow D+E
\end{aligned}
$$



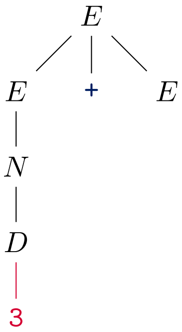
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow E+E*E \\
&\Rightarrow N+E*E \\
&\Rightarrow D+E*E
\end{aligned}
$$



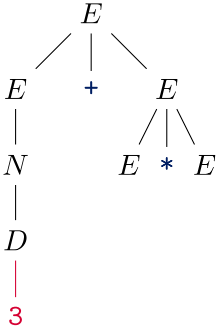
$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow N+E \\
&\Rightarrow D+E \\
&\Rightarrow 3+E
\end{aligned}
$$



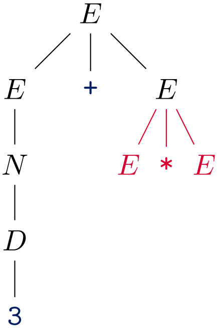
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow E+E*E \\ &\Rightarrow N+E*E \\ &\Rightarrow D+E*E \\ &\Rightarrow 3+E*E \end{aligned}$$



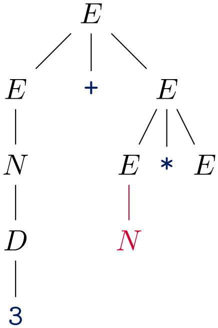
$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow N+E \\ &\Rightarrow D+E \\ &\Rightarrow 3+E \\ &\Rightarrow 3+E*E \end{aligned}$$



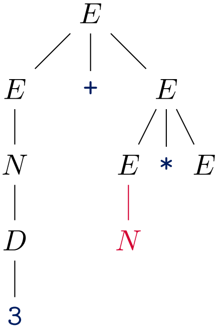
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$
\begin{aligned}
E &\Rightarrow E\texttt{+}E \\
&\Rightarrow E\texttt{+}E\ast E \\
&\Rightarrow N\texttt{+}E\ast E \\
&\Rightarrow D\texttt{+}E\ast E \\
&\Rightarrow \texttt{3+}E\ast E \\
&\Rightarrow \texttt{3+}N\ast E
\end{aligned}
$$



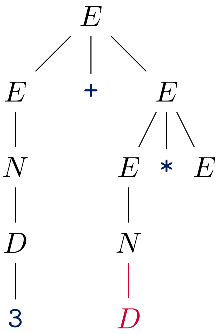
$$
\begin{aligned}
E &\Rightarrow E\texttt{+}E \\
&\Rightarrow N\texttt{+}E \\
&\Rightarrow D\texttt{+}E \\
&\Rightarrow \texttt{3+}E \\
&\Rightarrow \texttt{3+}E\ast E \\
&\Rightarrow \texttt{3+}N\ast E
\end{aligned}
$$



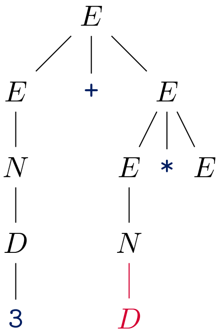
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$\begin{aligned} E &\Rightarrow E\text{+}E \\ &\Rightarrow E\text{+}E*E \\ &\Rightarrow N\text{+}E*E \\ &\Rightarrow D\text{+}E*E \\ &\Rightarrow 3\text{+}E*E \\ &\Rightarrow 3\text{+}N*E \\ &\Rightarrow 3\text{+}D*E \end{aligned}$$



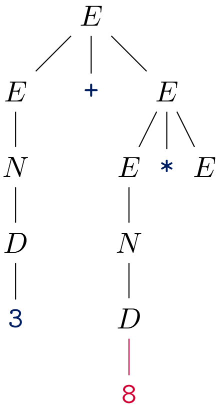
$$\begin{aligned} E &\Rightarrow E\text{+}E \\ &\Rightarrow N\text{+}E \\ &\Rightarrow D\text{+}E \\ &\Rightarrow 3\text{+}E \\ &\Rightarrow 3\text{+}E*E \\ &\Rightarrow 3\text{+}N*E \\ &\Rightarrow 3\text{+}D*E \end{aligned}$$



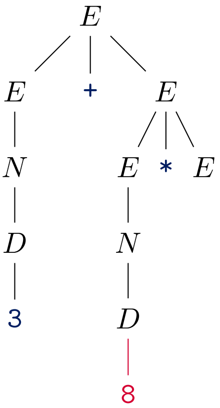
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$\begin{aligned} E &\Rightarrow E\text{+}E \\ &\Rightarrow E\text{+}E*E \\ &\Rightarrow N\text{+}E*E \\ &\Rightarrow D\text{+}E*E \\ &\Rightarrow 3\text{+}E*E \\ &\Rightarrow 3\text{+}N*E \\ &\Rightarrow 3\text{+}D*E \\ &\Rightarrow 3\text{+}8*E \end{aligned}$$



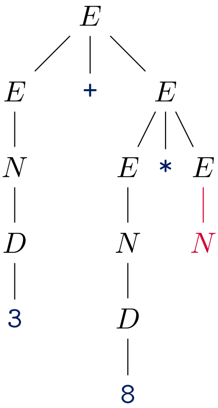
$$\begin{aligned} E &\Rightarrow E\text{+}E \\ &\Rightarrow N\text{+}E \\ &\Rightarrow D\text{+}E \\ &\Rightarrow 3\text{+}E \\ &\Rightarrow 3\text{+}E*E \\ &\Rightarrow 3\text{+}N*E \\ &\Rightarrow 3\text{+}D*E \\ &\Rightarrow 3\text{+}8*E \end{aligned}$$



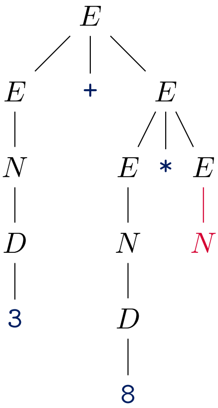
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow E+E*E \\
&\Rightarrow N+E*E \\
&\Rightarrow D+E*E \\
&\Rightarrow 3+E*E \\
&\Rightarrow 3+N*E \\
&\Rightarrow 3+D*E \\
&\Rightarrow 3+8*E \\
&\Rightarrow 3+8*N
\end{aligned}
$$



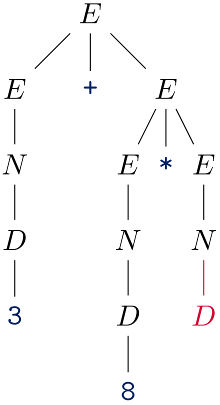
$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow N+E \\
&\Rightarrow D+E \\
&\Rightarrow 3+E \\
&\Rightarrow 3+E*E \\
&\Rightarrow 3+N*E \\
&\Rightarrow 3+D*E \\
&\Rightarrow 3+8*E \\
&\Rightarrow 3+8*N
\end{aligned}
$$



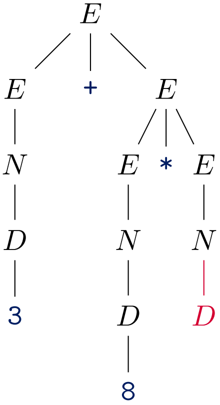
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow E+E*E \\ &\Rightarrow N+E*E \\ &\Rightarrow D+E*E \\ &\Rightarrow 3+E*E \\ &\Rightarrow 3+N*E \\ &\Rightarrow 3+D*E \\ &\Rightarrow 3+8*E \\ &\Rightarrow 3+8*N \\ &\Rightarrow 3+8*D \end{aligned}$$



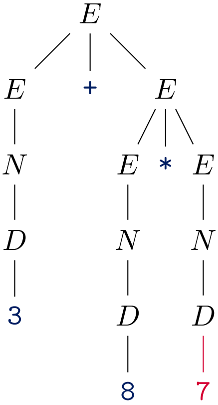
$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow N+E \\ &\Rightarrow D+E \\ &\Rightarrow 3+E \\ &\Rightarrow 3+E*E \\ &\Rightarrow 3+N*E \\ &\Rightarrow 3+D*E \\ &\Rightarrow 3+8*E \\ &\Rightarrow 3+8*N \\ &\Rightarrow 3+8*D \end{aligned}$$



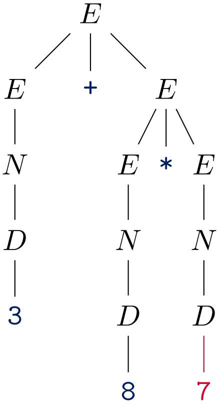
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow E+E*E \\
&\Rightarrow N+E*E \\
&\Rightarrow D+E*E \\
&\Rightarrow 3+E*E \\
&\Rightarrow 3+N*E \\
&\Rightarrow 3+D*E \\
&\Rightarrow 3+8*E \\
&\Rightarrow 3+8*N \\
&\Rightarrow 3+8*D \\
&\Rightarrow 3+8*7
\end{aligned}
$$



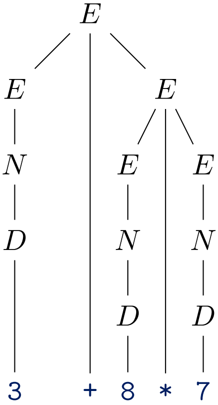
$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow N+E \\
&\Rightarrow D+E \\
&\Rightarrow 3+E \\
&\Rightarrow 3+E*E \\
&\Rightarrow 3+N*E \\
&\Rightarrow 3+D*E \\
&\Rightarrow 3+8*E \\
&\Rightarrow 3+8*N \\
&\Rightarrow 3+8*D \\
&\Rightarrow 3+8*7
\end{aligned}
$$



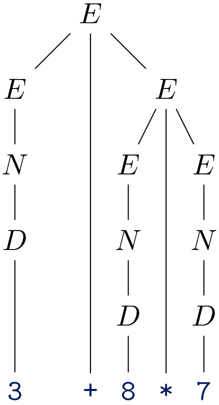
Different derivations can give rise to the same parse tree

Two different derivations give the same parse tree

$$
\begin{aligned}
E &\Rightarrow E{+}E \\
&\Rightarrow E{+}E{*}E \\
&\Rightarrow N{+}E{*}E \\
&\Rightarrow D{+}E{*}E \\
&\Rightarrow 3{+}E{*}E \\
&\Rightarrow 3{+}N{*}E \\
&\Rightarrow 3{+}D{*}E \\
&\Rightarrow 3{+}8{*}E \\
&\Rightarrow 3{+}8{*}N \\
&\Rightarrow 3{+}8{*}D \\
&\Rightarrow 3{+}8{*}7
\end{aligned}
$$



$$
\begin{aligned}
E &\Rightarrow E{+}E \\
&\Rightarrow N{+}E \\
&\Rightarrow D{+}E \\
&\Rightarrow 3{+}E \\
&\Rightarrow 3{+}E{*}E \\
&\Rightarrow 3{+}N{*}E \\
&\Rightarrow 3{+}D{*}E \\
&\Rightarrow 3{+}8{*}E \\
&\Rightarrow 3{+}8{*}N \\
&\Rightarrow 3{+}8{*}D \\
&\Rightarrow 3{+}8{*}7
\end{aligned}
$$

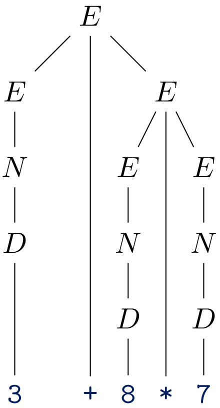


You can think of the derivations as filling out the tree in different orders

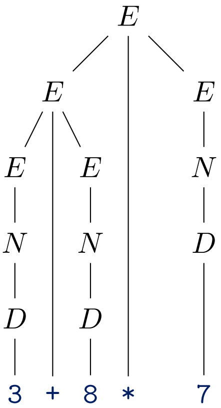
Different derivations can give rise to different parse trees

Two different left-most derivations give rise to different parse trees

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow N+E \\
&\Rightarrow D+E \\
&\Rightarrow 3+E \\
&\Rightarrow 3+E*E \\
&\Rightarrow 3+N*E \\
&\Rightarrow 3+D*E \\
&\Rightarrow 3+8*E \\
&\Rightarrow 3+8*N \\
&\Rightarrow 3+8*D \\
&\Rightarrow 3+8*7
\end{aligned}
$$

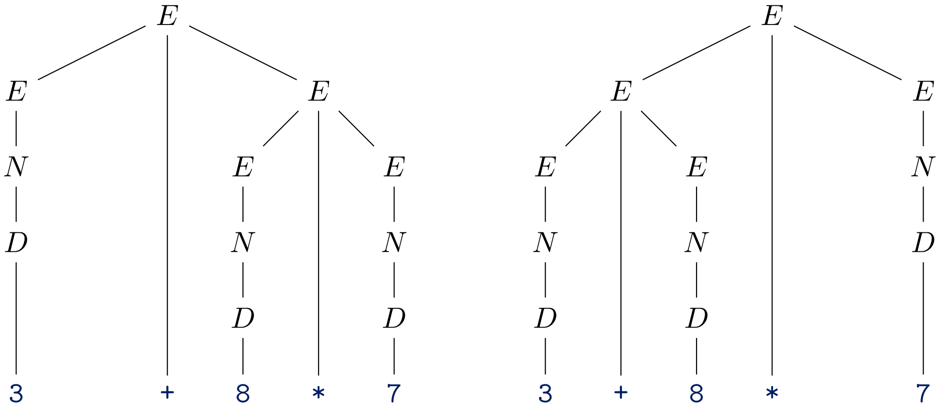


$$
\begin{aligned}
E &\Rightarrow E*E \\
&\Rightarrow E+E*E \\
&\Rightarrow N+E*E \\
&\Rightarrow D+E*E \\
&\Rightarrow 3+E*E \\
&\Rightarrow 3+N*E \\
&\Rightarrow 3+D*E \\
&\Rightarrow 3+8*E \\
&\Rightarrow 3+8*N \\
&\Rightarrow 3+8*D \\
&\Rightarrow 3+8*7
\end{aligned}
$$



Ambiguity

Our grammar can derive this string in two different ways



This grammar is ambiguous because it has two different parse trees for the same string in the language

Imagine a calculator or a compiler parsing this expression
Depending on which parse tree it used, it gets different results

Resolving ambiguity

In some cases, we can redesign the grammar to get rid of ambiguity

Instead of just expressions, let's have expressions (E), terms (T), and factors (F)

$$E \to E\texttt{+}T \mid E\texttt{-}T \mid T$$
$$T \to T\texttt{*}F \mid T\texttt{/}F \mid F$$
$$F \to \texttt{(}E\texttt{)} \mid N$$
$$N \to DN \mid D$$
$$D \to \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9}$$

This CFG has exactly the same language as the previous one but now there's exactly one way to parse 3+8*7

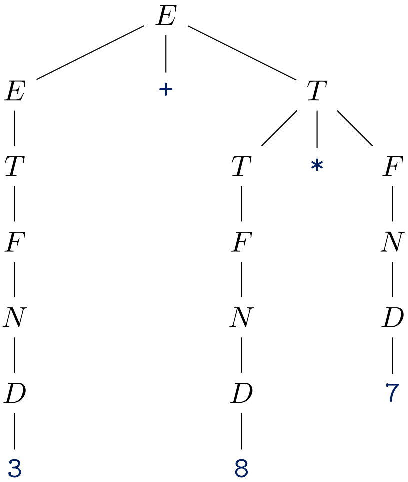
Resolving ambiguity

In some cases, we can redesign the grammar to get rid of ambiguity

Instead of just expressions, let's have expressions (E), terms (T), and factors (F)

$$E \to E\texttt{+}T \mid E\texttt{-}T \mid T$$
$$T \to T\texttt{*}F \mid T\texttt{/}F \mid F$$
$$F \to \texttt{(}E\texttt{)} \mid N$$
$$N \to DN \mid D$$
$$D \to \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9}$$

This CFG has exactly the same language as the previous one but now there's exactly one way to parse 3+8*7



Ambiguity

Equivalent statements about a CFG G

1. G is ambiguous if a word in $L(G)$ has two different parse trees
2. G is ambiguous if a word in $L(G)$ has two different left-most derivations
3. G is ambiguous if a word in $L(G)$ has two different right-most derivations

It is not the case that G is ambiguous if a word merely has two different derivations

Context-free languages

A language A is a context-free language (CFL) if there is a CFG G that generates A (i.e., $L(G) = A$)

Theorem

Context-free languages are closed under union, concatenation, and Kleene star.

Union

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A and $G_2 = (V_2, \Sigma, R_2, S_2)$ generate B (assume $V_1 \cap V_2 = \varnothing$, otherwise rename some variables)

Union

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A and $G_2 = (V_2, \Sigma, R_2, S_2)$ generate B (assume $V_1 \cap V_2 = \varnothing$, otherwise rename some variables)

Construct a new CFG $G = (V, \Sigma, R, S)$ to generate $A \cup B$ where

$$V = V_1 \cup V_2 \cup \{S\}$$
$$R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}$$

Union

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A and $G_2 = (V_2, \Sigma, R_2, S_2)$ generate B (assume $V_1 \cap V_2 = \varnothing$, otherwise rename some variables)

Construct a new CFG $G = (V, \Sigma, R, S)$ to generate $A \cup B$ where

$$V = V_1 \cup V_2 \cup \{S\}$$
$$R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}$$

If $w \in A$, then G_1 derives w, $S_1 \overset{*}{\Rightarrow} w$, and so G derives w via $S \Rightarrow S_1 \overset{*}{\Rightarrow} w$.

Union

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A and $G_2 = (V_2, \Sigma, R_2, S_2)$ generate B (assume $V_1 \cap V_2 = \varnothing$, otherwise rename some variables)

Construct a new CFG $G = (V, \Sigma, R, S)$ to generate $A \cup B$ where

$$V = V_1 \cup V_2 \cup \{S\}$$
$$R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}$$

If $w \in A$, then G_1 derives w, $S_1 \overset{*}{\Rightarrow} w$, and so G derives w via $S \Rightarrow S_1 \overset{*}{\Rightarrow} w$.
If $w \in B$, then $S_2 \overset{*}{\Rightarrow} w$ so $S \Rightarrow S_2 \overset{*}{\Rightarrow} w$. In either case $w \in L(G)$.

Union

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A and $G_2 = (V_2, \Sigma, R_2, S_2)$ generate B (assume $V_1 \cap V_2 = \varnothing$, otherwise rename some variables)

Construct a new CFG $G = (V, \Sigma, R, S)$ to generate $A \cup B$ where

$$V = V_1 \cup V_2 \cup \{S\}$$
$$R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}$$

If $w \in A$, then G_1 derives w, $S_1 \overset{*}{\Rightarrow} w$, and so G derives w via $S \Rightarrow S_1 \overset{*}{\Rightarrow} w$.
If $w \in B$, then $S_2 \overset{*}{\Rightarrow} w$ so $S \Rightarrow S_2 \overset{*}{\Rightarrow} w$. In either case $w \in L(G)$.

If $w \in L(G)$, then either $S \Rightarrow S_1 \overset{*}{\Rightarrow} w$ or $S \Rightarrow S_2 \overset{*}{\Rightarrow} w$. Thus $w \in A \cup B$. $\square$

Concatenation

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A and $G_2 = (V_2, \Sigma, R_2, S_2)$ generate B

Concatenation

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A and $G_2 = (V_2, \Sigma, R_2, S_2)$ generate B

Construct a new CFG $G = (V, \Sigma, R, S)$ to generate $A \circ B$ where

$$V = V_1 \cup V_2 \cup \{S\}$$
$$R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$$

A similar argument shows why $L(G) = A \circ B$ □

Kleene star

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A.

Kleene star

Proof.

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ generate A.

Construct a new CFG $G = (V, \Sigma, R, S)$ to generate A^* where

$$V = V_1 \cup \{S\}$$
$$R = R_1 \cup \{S \to SS_1 \mid \varepsilon\}$$

□

Regular languages are context-free

Theorem

Every regular language is context-free.

Regular languages are context-free

Theorem

Every regular language is context-free.

Proof.

We can use induction on the structure of regular expressions.

Three base cases

Regular languages are context-free

Theorem

Every regular language is context-free.

Proof.

We can use induction on the structure of regular expressions.

Three base cases

- $\underline{\varnothing}$. $S \rightarrow S$

Regular languages are context-free

Theorem

Every regular language is context-free.

Proof.

We can use induction on the structure of regular expressions.

Three base cases

- $\underline{\varnothing}$. $S \to S$
- $\underline{\varepsilon}$. $S \to \varepsilon$

Regular languages are context-free

Theorem

Every regular language is context-free.

Proof.

We can use induction on the structure of regular expressions.

Three base cases

- $\underline{\emptyset}$. $S \to S$
- $\underline{\varepsilon}$. $S \to \varepsilon$
- $\underline{t}$ for $t \in \Sigma$. $S \to t$

Regular languages are context-free

Theorem

Every regular language is context-free.

Proof.

We can use induction on the structure of regular expressions.

Three base cases

- $\underline{\varnothing}$. $S \to S$
- $\underline{\varepsilon}$. $S \to \varepsilon$
- $\underline{t}$ for $t \in \Sigma$. $S \to t$

Three inductive cases.

- $\underline{R_1 R_2}$
- $\underline{R_1 \mid R_2}$
- $\underline{R_1^*}$

Regular languages are context-free

Theorem

Every regular language is context-free.

Proof.

We can use induction on the structure of regular expressions.

Three base cases

- $\underline{\varnothing}$. $S \to S$
- $\underline{\varepsilon}$. $S \to \varepsilon$
- $\underline{t}$ for $t \in \Sigma$. $S \to t$

Three inductive cases.

- $\underline{R_1 R_2}$
- $\underline{R_1 \mid R_2}$
- $\underline{R_1^*}$

By the inductive hypothesis, $L(R_1)$ and $L(R_2)$ are context-free and context-free languages are closed under concatenation, union, and star. □

Ambiguity

An inherently ambiguous context-free language is one in which every context-free grammar is ambiguous

$\{a^i b^j c^k \mid i = j \text{ or } j = k\}$ is inherently ambiguous