02141 Computer Science Modelling
Context-Free Languages

CFL1: Introduction to Context Free Grammars

Why should we learn about grammars?

Why should we learn about grammars?

Why should we learn about grammars?

Why should we learn about grammars?

- To describe the **syntax of programming languages** in a precise way.

Why should we learn about grammars?

- To describe the **syntax of programming languages** in a precise way.

  Applications: analysers, optimisers, compilers, interpreters, etc.

Why should we learn about grammars?

- To describe the **syntax of programming languages** in a precise way.

  Applications: analysers, optimisers, compilers, interpreters, etc.

- To develop **data formats** that can be efficiently manipulated.

Why should we learn about grammars?

- To describe the **syntax of programming languages** in a precise way.

  Applications: analysers, optimisers, compilers, interpreters, etc.

- To develop **data formats** that can be efficiently manipulated.

  Applications: XML, JSON, etc.

A Simple Programming Language

$$
\begin{array}{lcl}
PROG & \rightarrow & STMT \; ; \\
PROG & \rightarrow & \{STMTS\} \\
STMTS & \rightarrow & STMT ; \; STMTS \\
STMTS & \rightarrow & \epsilon \\
 & & \\
STMT & \rightarrow & VAR = EXP \\
STMT & \rightarrow & \mathit{if} \; (COND) \; PROG \\
STMT & \rightarrow & \mathit{if} \; (COND) \; PROG \; \mathit{else} \; PROG \\
STMT & \rightarrow & \mathit{while} \; (COND) \; PROG \\
 & & \\
EXP & \rightarrow & VAR \\
EXP & \rightarrow & CONST \\
EXP & \rightarrow & EXP + EXP \\
EXP & \rightarrow & EXP * EXP \\
EXP & \rightarrow & (EXP) \\
\ldots & &
\end{array}
$$

Some “syntactic” aspects of programming languages cannot be expressed by a context-free language, for instance:

- that all variables have been declared
- that the program is type-correct

The Chomsky Hierarchy of Grammars

Language	Machine	Grammar	Productions
Regular	Finite-state automaton	Regular	$A \to aB$ $A \to \epsilon$
Context-free	Pushdown automaton	Context-free	$A \to \gamma$
Context-sensitive	Linear-bounded Turing machine	Context-sensitive	$\alpha A \beta \to \gamma$
Recursively enumerable	Turing machine	Unrestricted	$\alpha \to \gamma$

The Chomsky Hierarchy of Grammars

Language	Machine	Grammar	Productions
Regular	Finite-state automaton	Regular	$A \to aB$ $A \to \epsilon$
Context-free	Pushdown automaton	Context-free	$A \to \gamma$
Context-sensitive	Linear-bounded Turing machine	Context-sensitive	$\alpha A \beta \to \gamma$
Recursively enumerable	Turing machine	Unrestricted	$\alpha \to \gamma$

Part I of the course was about **regular languages**.

The Chomsky Hierarchy of Grammars

Language	Machine	Grammar	Productions
Regular	Finite-state automaton	Regular	$A \to aB$ $A \to \epsilon$
Context-free	Pushdown automaton	Context-free	$A \to \gamma$
Context-sensitive	Linear-bounded Turing machine	Context-sensitive	$\alpha A \beta \to \gamma$
Recursively enumerable	Turing machine	Unrestricted	$\alpha \to \gamma$

This part of the course is about **context-free languages** — loosely speaking, the **syntax** of programming languages.

What is a Grammar?

A **grammar** defines a language using a set of rules, called **productions**:

$$\alpha \to \gamma$$

What is a Grammar?

A **grammar** defines a language using a set of rules, called **productions**:

$$\alpha \rightarrow \gamma$$

α (the head) and γ (the body) are sequences of **symbols**, of which there are two types:

What is a Grammar?

A **grammar** defines a language using a set of rules, called **productions**:

$$\alpha \to \gamma$$

α (the head) and γ (the body) are sequences of **symbols**, of which there are two types:

- **Terminals** are the symbols in the alphabet of the language. They are usually written in lowercase.

What is a Grammar?

A **grammar** defines a language using a set of rules, called **productions**:

$$\alpha \rightarrow \gamma$$

α (the head) and γ (the body) are sequences of **symbols**, of which there are two types:

- **Terminals** are the symbols in the alphabet of the language. They are usually written in lowercase.
- **Variables** (also called **non-terminals** or **syntactic categories**) are symbols outside the alphabet of the language. They are usually written in uppercase.

What is a Grammar?

A **grammar** defines a language using a set of rules, called **productions**:

$$A \to \gamma$$

α (the head) and γ (the body) are sequences of **symbols**, of which there are two types:

- **Terminals** are the symbols in the alphabet of the language. They are usually written in lowercase.
- **Variables** (also called **non-terminals** or **syntactic categories**) are symbols outside the alphabet of the language. They are usually written in uppercase.

In a **context-free grammar**, the left-hand side of a production is always a single variable.

What is a Grammar?

A **grammar** defines a language using a set of rules, called **productions**:

$$A \rightarrow \gamma$$

α (the head) and γ (the body) are sequences of **symbols**, of which there are two types:

- **Terminals** are the symbols in the alphabet of the language. They are usually written in lowercase.
- **Variables** (also called **non-terminals** or **syntactic categories**) are symbols outside the alphabet of the language. They are usually written in uppercase.

In a **context-free grammar**, the left-hand side of a production is always a single variable.

This tells us that we can replace A by γ — starting with a variable, we can keep on replacing variables until we end up with a string of terminals!

What is a Grammar?

Example: a grammar describing the language of palindromes over $\{0, 1\}$:

$$\begin{array}{lcl} P & \rightarrow & \epsilon \\ P & \rightarrow & 0 \\ P & \rightarrow & 1 \end{array} \qquad \begin{array}{lcl} P & \rightarrow & 0P0 \\ P & \rightarrow & 1P1 \end{array}$$

What is a Grammar?

Example: a grammar describing the language of palindromes over $\{0, 1\}$:

$$\begin{array}{lllllll} P & \rightarrow & \epsilon & \qquad & P & \rightarrow & 0P0 \\ P & \rightarrow & 0 & & P & \rightarrow & 1P1 \\ P & \rightarrow & 1 & & & & \end{array}$$

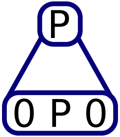
P

What is a Grammar?

Example: a grammar describing the language of palindromes over $\{0, 1\}$:

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
P & \rightarrow & \epsilon & P & \rightarrow & 0P0 \\
P & \rightarrow & 0 & P & \rightarrow & 1P1 \\
P & \rightarrow & 1 & & &
\end{array}
$$

$\Rightarrow$



What is a Grammar?

Example: a grammar describing the language of palindromes over $\{0, 1\}$:

$$
\begin{array}{lclclcl}
P & \rightarrow & \epsilon & \qquad & P & \rightarrow & 0P0 \\
P & \rightarrow & 0 & & P & \rightarrow & 1P1 \\
P & \rightarrow & 1 & & & &
\end{array}
$$

$$
\begin{array}{cc}
 & \mathrm{P} \\
\Rightarrow & 0\,\boxed{\mathrm{P}}\,0 \\
\Rightarrow & 0\,\boxed{1\ \mathrm{P}\ 1}\,0
\end{array}
$$

What is a Grammar?

Example: a grammar describing the language of palindromes over $\{0, 1\}$:

$$
\begin{array}{lcllcl}
P & \to & \epsilon & P & \to & 0P0 \\
P & \to & 0 & P & \to & 1P1 \\
P & \to & 1 & & &
\end{array}
$$

$$
\begin{array}{lc}
 & P \\
\Rightarrow & 0\ P\ 0 \\
\Rightarrow & 0\ 1\ P\ 1\ 0 \\
\Rightarrow & 0\ 1\ 0\ P\ 0\ 1\ 0
\end{array}
$$

What is a Grammar?

Example: a grammar describing the language of palindromes over $\{0, 1\}$:

$$\begin{array}{lcl} P & \rightarrow & \epsilon \\ P & \rightarrow & 0 \\ P & \rightarrow & 1 \end{array} \qquad \begin{array}{lcl} P & \rightarrow & 0P0 \\ P & \rightarrow & 1P1 \end{array}$$

$$\begin{array}{ll} & \mathrm{P} \\ \Rightarrow & 0\ \mathrm{P}\ 0 \\ \Rightarrow & 0\ 1\ \mathrm{P}\ 1\ 0 \\ \Rightarrow & 0\ 1\ 0\ \mathrm{P}\ 0\ 1\ 0 \\ \Rightarrow & 0\ 1\ 0\ 0\ 0\ 1\ 0 \end{array}$$

What is a Grammar?

Example: a grammar describing the language of palindromes over $\{0, 1\}$:

$$
\begin{array}{lllllll}
P & \rightarrow & \epsilon & \quad & P & \rightarrow & 0P0 \\
P & \rightarrow & 0 & & P & \rightarrow & 1P1 \\
P & \rightarrow & 1 & & & &
\end{array}
$$

$$
\begin{array}{ll}
 & \mathrm{P} \\
\Rightarrow & 0\ \mathrm{P}\ 0 \\
\Rightarrow & 0\ 1\ \mathrm{P}\ 1\ 0 \\
\Rightarrow & 0\ 1\ 0\ \mathrm{P}\ 0\ 1\ 0 \\
\Rightarrow & 0\ 1\ 0\ 0\ 0\ 1\ 0
\end{array}
$$

Context-Free Grammars

Context-Free Grammars

A **context-free grammar** is a quadruple $G = (V, T, P, S)$ where:

Context-Free Grammars

A **context-free grammar** is a quadruple $G = (V, T, P, S)$ where:

- V is a finite set of symbols called the **variables** or **non-terminals**.

Context-Free Grammars

A **context-free grammar** is a quadruple $G = (V, T, P, S)$ where:

- V is a finite set of symbols called the **variables** or **non-terminals**.
- T is a finite set of symbols, disjoint from V, called the **terminals**.

Context-Free Grammars

A **context-free grammar** is a quadruple $G = (V, T, P, S)$ where:

- V is a finite set of symbols called the **variables** or **non-terminals**.
- T is a finite set of symbols, disjoint from V, called the **terminals**.
- $P \subseteq V \times (V \cup T)^\star$ is a finite set of **productions**.

  where a rule $(A, \alpha) \in P$ is written $A \to \alpha$.

Context-Free Grammars

A **context-free grammar** is a quadruple $G = (V, T, P, S)$ where:

- V is a finite set of symbols called the **variables** or **non-terminals**.
- T is a finite set of symbols, disjoint from V, called the **terminals**.
- $P \subseteq V \times (V \cup T)^{\star}$ is a finite set of **productions**.
  where a rule $(A, \alpha) \in P$ is written $A \rightarrow \alpha$.
- $S \in V$ is the **start symbol**.

Context-Free Grammars

A **context-free grammar** is a quadruple $G = (V, T, P, S)$ where:

- V is a finite set of symbols called the **variables** or **non-terminals**.
- T is a finite set of symbols, disjoint from V, called the **terminals**.
- $P \subseteq V \times (V \cup T)^{\star}$ is a finite set of **productions**.
  where a rule $(A, \alpha) \in P$ is written $A \to \alpha$.
- $S \in V$ is the **start symbol**.

If we have a set of rules $A \to \alpha_1, \ldots, A \to \alpha_n$, we use the short-hand:

$$A \to \alpha_1 \mid \cdots \mid \alpha_n$$

Exercise 1.1

Design context-free grammars for the following languages:
a) $\{0^n \mid n \geq 0\}$, i.e. the set of all strings made of zero or more 0's.
b) $\{0^n \mid n \geq 1\}$, i.e. the set of all strings made of one or more 0's.
c) $\{0^n 1^m \mid n \geq 1 \wedge m \geq 1\}$, i.e. the set of all strings made of one or more 0's followed by one or more 1's.
d) $\{0^n 1^n \mid n \geq 1\}$, i.e. the set of all strings made of one or more 0's followed by an equal number of 1's.

NOTE: n and m are natural numbers.

The Language of a Context-Free Grammar

- Given a string $\alpha B\gamma$ and a grammar rule $B \rightarrow \beta$, we can derive $\alpha\beta\gamma$. We write this as:

$$\alpha B\gamma \Rightarrow \alpha\beta\gamma$$

- $\stackrel{\star}{\Rightarrow}$ is the reflexive, transitive closure of $\Rightarrow$

  i.e. an arbitrary sequence of derivation steps.
- The language $L(G)$ of grammar $G = (V, T, P, S)$ is defined as

$$L(G) = \{w \in T^* \mid S \stackrel{\star}{\Rightarrow} w\}$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$\begin{array}{lclclclcl} E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclclclcl}
E & \to & N & \mid & E+E & \mid & E\times E & \mid & (E) \\
N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$E \quad \Rightarrow \quad E \times E \qquad\qquad \text{using } E \to E \times E$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclclclcl}
E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{lcll}
E & \Rightarrow & E \times E & \text{using } E \rightarrow E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \rightarrow (E)
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$\begin{array}{lcccccccc} E & \rightarrow & N & \mid & E+E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

$$\begin{array}{lcll} E & \Rightarrow & E \times E & \text{using } E \rightarrow E \times E \\ & \Rightarrow & (E) \times E & \text{using } E \rightarrow (E) \\ & \Rightarrow & ({\color{red}E + E}) \times E & \text{using } E \rightarrow E + E \end{array}$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclcccccc}
E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{lcll}
E & \Rightarrow & E \times E & \text{using } E \rightarrow E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \rightarrow (E) \\
 & \Rightarrow & (E + E) \times E & \text{using } E \rightarrow E + E \\
 & \Rightarrow & (E + \color{red}{N}) \times E & \text{using } E \rightarrow N
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclcccccc}
E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{lcll}
E & \Rightarrow & E \times E & \text{using } E \rightarrow E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \rightarrow (E) \\
 & \Rightarrow & (E + E) \times E & \text{using } E \rightarrow E + E \\
 & \Rightarrow & (E + N) \times E & \text{using } E \rightarrow N \\
 & \Rightarrow & (E + 1) \times E & \text{using } N \rightarrow 1
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclcccccc}
E & \to & N & \mid & E+E & \mid & E\times E & \mid & (E) \\
N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{lcll}
E & \Rightarrow & E \times E & \text{using } E \to E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \to (E) \\
 & \Rightarrow & (E+E) \times E & \text{using } E \to E + E \\
 & \Rightarrow & (E+N) \times E & \text{using } E \to N \\
 & \Rightarrow & (E+1) \times E & \text{using } N \to 1 \\
 & \Rightarrow & (\color{red}{N}+1) \times E & \text{using } E \to N
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lllllllll}
E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{llll}
E & \Rightarrow & E \times E & \text{using } E \rightarrow E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \rightarrow (E) \\
 & \Rightarrow & (E + E) \times E & \text{using } E \rightarrow E + E \\
 & \Rightarrow & (E + N) \times E & \text{using } E \rightarrow N \\
 & \Rightarrow & (E + 1) \times E & \text{using } N \rightarrow 1 \\
 & \Rightarrow & (N + 1) \times E & \text{using } E \rightarrow N \\
 & \Rightarrow & (N1 + 1) \times E & \text{using } N \rightarrow N1
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclclclcl}
E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{lcll}
E & \Rightarrow & E \times E & \text{using } E \rightarrow E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \rightarrow (E) \\
 & \Rightarrow & (E + E) \times E & \text{using } E \rightarrow E + E \\
 & \Rightarrow & (E + N) \times E & \text{using } E \rightarrow N \\
 & \Rightarrow & (E + 1) \times E & \text{using } N \rightarrow 1 \\
 & \Rightarrow & (N + 1) \times E & \text{using } E \rightarrow N \\
 & \Rightarrow & (N1 + 1) \times E & \text{using } N \rightarrow N1 \\
 & \Rightarrow & (11 + 1) \times E & \text{using } N \rightarrow 1
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclclclcl}
E & \to & N & \mid & E+E & \mid & E \times E & \mid & (E) \\
N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{lcll}
E & \Rightarrow & E \times E & \text{using } E \to E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \to (E) \\
 & \Rightarrow & (E+E) \times E & \text{using } E \to E+E \\
 & \Rightarrow & (E+N) \times E & \text{using } E \to N \\
 & \Rightarrow & (E+1) \times E & \text{using } N \to 1 \\
 & \Rightarrow & (N+1) \times E & \text{using } E \to N \\
 & \Rightarrow & (N1+1) \times E & \text{using } N \to N1 \\
 & \Rightarrow & (11+1) \times E & \text{using } N \to 1 \\
 & \Rightarrow & (11+1) \times N & \text{using } E \to N
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$
\begin{array}{lclcccccc}
E & \to & N & \mid & E+E & \mid & E \times E & \mid & (E) \\
N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

$$
\begin{array}{lcll}
E & \Rightarrow & E \times E & \text{using } E \to E \times E \\
 & \Rightarrow & (E) \times E & \text{using } E \to (E) \\
 & \Rightarrow & (E+E) \times E & \text{using } E \to E+E \\
 & \Rightarrow & (E+N) \times E & \text{using } E \to N \\
 & \Rightarrow & (E+1) \times E & \text{using } N \to 1 \\
 & \Rightarrow & (N+1) \times E & \text{using } E \to N \\
 & \Rightarrow & (N1+1) \times E & \text{using } N \to N1 \\
 & \Rightarrow & (11+1) \times E & \text{using } N \to 1 \\
 & \Rightarrow & (11+1) \times N & \text{using } E \to N \\
 & \Rightarrow & (11+1) \times N1 & \text{using } N \to N1
\end{array}
$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$\begin{array}{lcccccccc} E & \rightarrow & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

$$\begin{array}{llll} E & \Rightarrow & E\times E & \text{using } E\rightarrow E\times E \\ & \Rightarrow & (E)\times E & \text{using } E\rightarrow (E) \\ & \Rightarrow & (E+E)\times E & \text{using } E\rightarrow E+E \\ & \Rightarrow & (E+N)\times E & \text{using } E\rightarrow N \\ & \Rightarrow & (E+1)\times E & \text{using } N\rightarrow 1 \\ & \Rightarrow & (N+1)\times E & \text{using } E\rightarrow N \\ & \Rightarrow & (N1+1)\times E & \text{using } N\rightarrow N1 \\ & \Rightarrow & (11+1)\times E & \text{using } N\rightarrow 1 \\ & \Rightarrow & (11+1)\times N & \text{using } E\rightarrow N \\ & \Rightarrow & (11+1)\times N1 & \text{using } N\rightarrow N1 \\ & \Rightarrow & (11+1)\times 11 & \text{using } N\rightarrow 1 \end{array}$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$\begin{array}{lclclclcl} E & \to & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

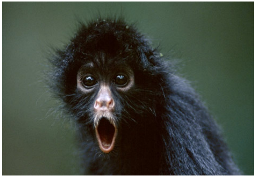
$$E \overset{\star}{\Rightarrow} (11 + 1) \times 11$$

Example

Consider the following grammar, for simple binary arithmetic expressions:

$$\begin{array}{lclclclcl} E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

$$E \overset{\star}{\Rightarrow} (11 + 1) \times 11$$



This is a **top-down** approach.

Left-most and Right-most Derivations

$$\begin{tikzcd}
 & E \arrow[d, Rightarrow] & \\
 & E \times E \arrow[dl, Rightarrow, "lm"'] \arrow[dr, Rightarrow, "rm"] & \\
(E) \times E \arrow[dr, Rightarrow, "\star", "lm"'] & & E \times N \arrow[dl, Rightarrow, "\star"', "rm"] \\
 & (11 + 1) \times 11 &
\end{tikzcd}$$

The derivation of a word is in general not unique

Left-most and Right-most Derivations

$$
\begin{tikzcd}
 & E \arrow[d, Rightarrow] & \\
 & {\color{blue}E} \times {\color{red}E} \arrow[dl, Rightarrow, "\color{blue}lm"'] \arrow[dr, Rightarrow, "\color{red}rm"] & \\
{\color{blue}(E)} \times E \arrow[dr, Rightarrow, "\star", "\color{blue}lm"'] & & E \times {\color{red}N} \arrow[dl, Rightarrow, "\star"', "\color{red}rm"] \\
 & (11+1) \times 11 &
\end{tikzcd}
$$

$\underset{lm}{\Rightarrow}$: derivation on the left-most variable symbol

$\underset{rm}{\Rightarrow}$: derivation on the right-most variable symbol

Left-most and Right-most Derivations

$$\begin{tikzcd}
& E \arrow[d, Rightarrow] & \\
& E \times E \arrow[dl, Rightarrow, "lm"'] \arrow[dr, Rightarrow, "rm"] & \\
(E) \times E \arrow[dr, Rightarrow, "\star", "lm"'] & & E \times N \arrow[dl, Rightarrow, "\star"', "rm"] \\
& (11 + 1) \times 11 &
\end{tikzcd}$$

It does not really matter:

Theorem. $S \overset{\star}{\Rightarrow} w$ iff $S \underset{lm}{\overset{\star}{\Rightarrow}} w$ iff $S \underset{rm}{\overset{\star}{\Rightarrow}} w$

Left-most and Right-most Derivations

$$
\begin{tikzcd}
 & E \arrow[d, Rightarrow] & \\
 & E \times E \arrow[dl, Rightarrow, "lm"'] \arrow[dr, Rightarrow, "rm"] & \\
 (E) \times E \arrow[dr, Rightarrow, "\star", "lm"'] & & E \times N \arrow[dl, Rightarrow, "\star"', "rm"] \\
 & (11+1) \times 11 &
\end{tikzcd}
$$

It does not really matter:

Theorem. $S \overset{\star}{\Rightarrow} w$ iff $S \underset{lm}{\overset{\star}{\Rightarrow}} w$ iff $S \underset{rm}{\overset{\star}{\Rightarrow}} w$

This is why these grammars are called **context-free** in the first place!

Exercise 1.2

Do the following exercise from the book:

Exercise 5.1.2: The following grammar generates the language of regular expression $\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^*$:

$$
\begin{array}{lcl}
S & \rightarrow & A1B \\
A & \rightarrow & 0A \mid \epsilon \\
B & \rightarrow & 0B \mid 1B \mid \epsilon
\end{array}
$$

Give leftmost and rightmost derivations of the following strings:

* a) 00101.

b) 1001.

Inductive Definition

$$\begin{array}{lclclclcl} E & \to & N & \mid & E+E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Compare with the commonly used **inductive definitions** such as:

Binary numbers are defined as follows:

- 0 is a binary number.
- 1 is a binary number.
- If N is a binary number, then $N0$ and $N1$ are binary numbers.
- Nothing else.

Expressions are defined as follows:

- If N is a binary number, then N is an expression.
- If E_1 and E_2 are expressions, then also $E_1 + E_2$, $E_1 \times E_2$, and (E_1) are expressions.
- Nothing else.

Inductive Definition—Recursive Inference

$$\begin{array}{lclclclclc} E & \to & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Inductive Definition—Recursive Inference

$$\begin{array}{rclcccccc} E & \rightarrow & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

Strings inferred for N:
Strings inferred for E:

Inductive Definition—Recursive Inference

$$\begin{array}{lcccccccc} E & \to & N & \mid & E+E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body

Strings inferred for N:
Strings inferred for E:

Inductive Definition—Recursive Inference

$$\begin{array}{lcccccccc} E & \rightarrow & N & \mid & E+E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body

Example: $0 \in L(N)$, since $N \rightarrow 0$

Strings inferred for N: 0
Strings inferred for E:

Inductive Definition—Recursive Inference

$$\begin{array}{lclcccccc} E & \rightarrow & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body

Example: $1 \in L(N)$, since $N \rightarrow 1$

Strings inferred for N: 0, 1
Strings inferred for E:

Inductive Definition—Recursive Inference

$$\begin{array}{lclcccccccc} E & \to & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

Strings inferred for N: 0, 1
Strings inferred for E:

Inductive Definition—Recursive Inference

$$\begin{array}{lllllllll} E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

Example: $11 \in L(N)$, since $N \rightarrow N1$ and $1 \in L(N)$

Strings inferred for N: 0, 1, 11
Strings inferred for E:

Inductive Definition—Recursive Inference

$$\begin{array}{lclcccccccc} E & \to & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

Example: $1 \in L(E)$, since $E \to N$ and $1 \in L(N)$

Strings inferred for N: 0, 1, 11
Strings inferred for E: 1

Inductive Definition—Recursive Inference

$$\begin{array}{lclcccccc} E & \rightarrow & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

Example: $11 \in L(E)$, since $E \rightarrow N$ and $11 \in L(N)$

Strings inferred for N: 0, 1, 11
Strings inferred for E: 1, 11

Inductive Definition—Recursive Inference

$$\begin{array}{lcccccccc} E & \to & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

Example: $11+1 \in L(E)$, since $E \to E+E$ and $11, 1 \in L(E)$

Strings inferred for N: 0, 1, 11
Strings inferred for E: 1, 11, $11+1$

Inductive Definition—Recursive Inference

$$\begin{array}{lclcccccc} E & \rightarrow & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

Example: $(11+1) \in L(E)$, since $E \rightarrow (E)$ and $11+1 \in L(E)$

Strings inferred for N: 0, 1, 11
Strings inferred for E: 1, 11, $11+1$, $(11+1)$

Inductive Definition—Recursive Inference

$$\begin{array}{lcccccccc} E & \to & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

Example: $(11 + 1) \times 11 \in L(E)$, since $E \to E \times E$ and $(11 + 1), 11 \in L(E)$

Strings inferred for N: 0, 1, 11
Strings inferred for E: 1, 11, $11 + 1$, $(11 + 1)$, $(11 + 1) \times 11$

Inductive Definition—Recursive Inference

$$\begin{array}{lllllllll} E & \to & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

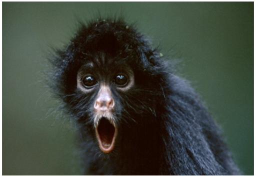
Following the idea of the inductive definition, we can recursively infer words of the languages $L(E)$ and $L(N)$:

- **Base case**:
  Rules with no variables in the body
- **Inductive case**:
  Rules with variables in the body. Variables can be replaced with strings that have already been inferred.

This is a **bottom-up** approach.

Derivation vs. Recursive Inference

We have seen two approaches to define the language of a context-free grammar

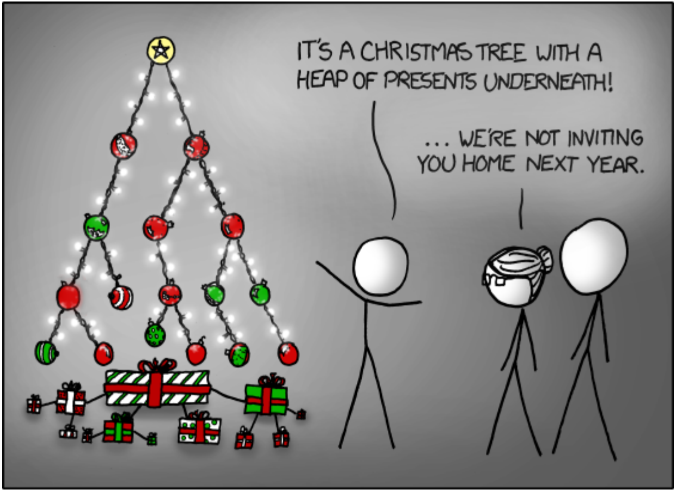


Top-down by derivation

Bottom-up by recursive inference

Theorem: Both definitions are equivalent.

Parse Trees



http://xkcd.com/835/

Parse Trees

In case you actually want to do something with an input word...

$$\begin{array}{lclcccccc} E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

A Parse Tree is any tree created as follows:

Parse Trees

In case you actually want to do something with an input word...

$$
\begin{array}{rcccccccc}
E & \to & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

A Parse Tree is any tree created as follows:

- Start with a single **root node**, labeled by start symbol S.

Parse Trees

In case you actually want to do something with an input word...

$$\begin{array}{lclclclcl} E & \rightarrow & N & \mid & E+E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

A Parse Tree is any tree created as follows:

- Start with a single **root node**, labeled by start symbol S.
- If a node is labeled by a variable A, choose a production $A \rightarrow X_1 \cdots X_n$, and create n **child nodes** labeled (from left to right) by $X_1 \cdots X_n$.
  (In case of a production $A \rightarrow \epsilon$, use one child node labeled ϵ.)

Parse Trees

In case you actually want to do something with an input word...

$$\begin{array}{rcccccccc} E & \to & N & \mid & E+E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

A Parse Tree is any tree created as follows:

- Start with a single **root node**, labeled by start symbol S.
- If a node is labeled by a variable A, choose a production $A \to X_1 \cdots X_n$, and create n **child nodes** labeled (from left to right) by $X_1 \cdots X_n$.
  (In case of a production $A \to \epsilon$, use one child node labeled ϵ.)
- Continue until all variable-labeled nodes have appropriate child nodes.

Parse Trees

In case you actually want to do something with an input word...

$$
\begin{array}{lclcccccc}
E & \rightarrow & N & \mid & E+E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

A Parse Tree is any tree created as follows:

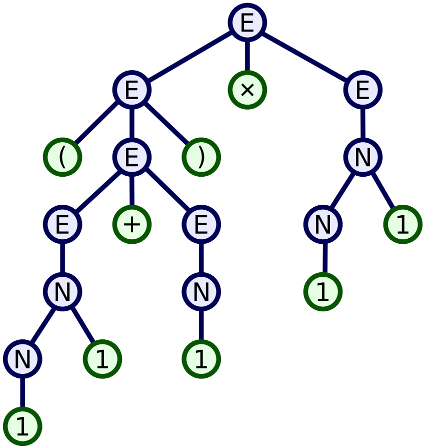
- Start with a single **root node**, labeled by start symbol S.
- If a node is labeled by a variable A, choose a production $A \rightarrow X_1 \cdots X_n$, and create n **child nodes** labeled (from left to right) by $X_1 \cdots X_n$.
  (In case of a production $A \rightarrow \epsilon$, use one child node labeled ϵ.)
- Continue until all variable-labeled nodes have appropriate child nodes.
- So every leaf node is labeled by a terminal or by ϵ.

Parse Trees

In case you actually want to do something with an input word...

$$
\begin{array}{lcccccccc}
E & \rightarrow & N & \mid & E+E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

A Parse Tree is any tree created as follows:

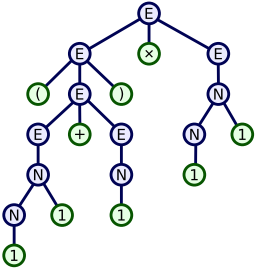


Parse Trees

In case you actually want to do something with an input word...

$$\begin{array}{lcccccccc} E & \to & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

A Parse Tree is any tree created as follows:



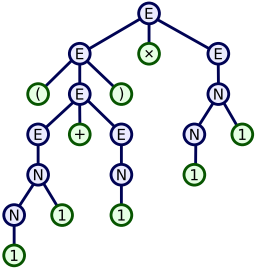
The **yield** of a parse tree is the string obtained by concatenating the leaves in a depth-first traversal of the tree.

Parse Trees

In case you actually want to do something with an input word...

$$
\begin{array}{lclclclcl}
E & \to & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

A Parse Tree is any tree created as follows:



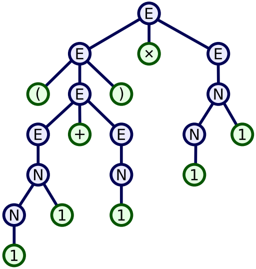
The **yield** of a parse tree is the string obtained by concatenating the leaves in a depth-first traversal of the tree.
Yield of the example: $(11 + 1) \times 11$

Parse Trees

In case you actually want to do something with an input word...

$$\begin{array}{rcccccccc} E & \rightarrow & N & \mid & E+E & \mid & E\times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

A Parse Tree is any tree created as follows:



The **yield** of a parse tree is the string obtained by concatenating the leaves in a depth-first traversal of the tree.

Theorem: $w \in L(G)$ iff there is a parse tree with yield w.

The Language of a Context-Free Grammar

$$\begin{array}{rcccccccc} E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Is a parse tree always unique?

The Language of a Context-Free Grammar

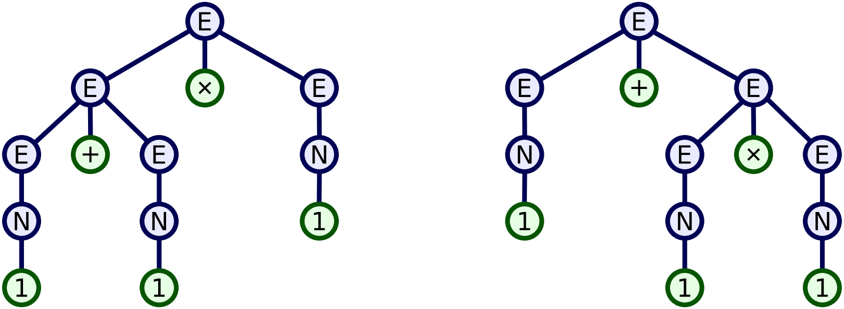
$$\begin{array}{lcccccccc} E & \to & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Is a parse tree always unique? NO

The Language of a Context-Free Grammar

$$\begin{array}{lcccccccc} E & \to & N & \mid & E+E & \mid & E \times E & \mid & (E) \\ N & \to & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

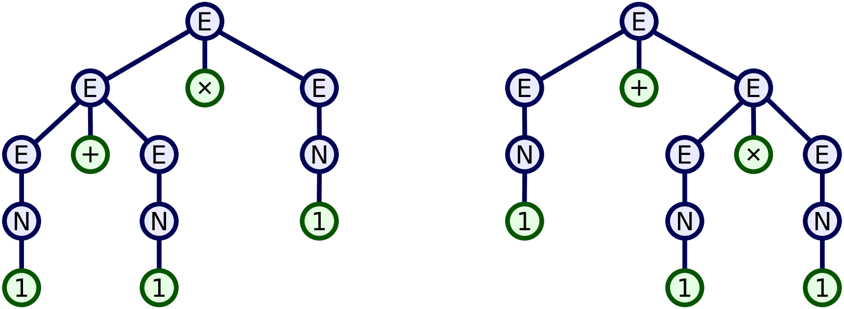
Is a parse tree always unique? NO



The Language of a Context-Free Grammar

$$\begin{array}{lclclclcl} E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\ N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1 \end{array}$$

Is a parse tree always unique? NO

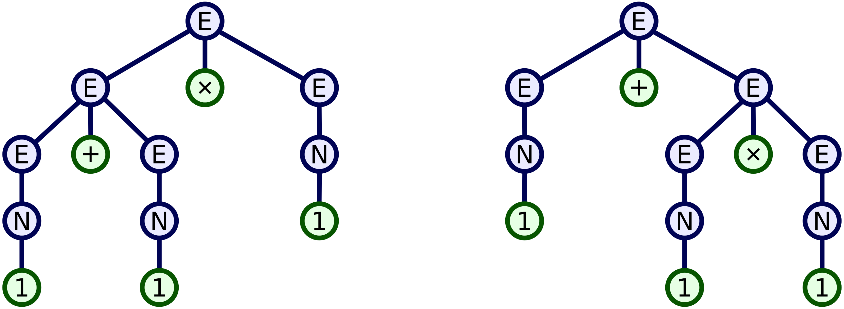


Both parse trees yield the string $1 + 1 \times 1$.

The Language of a Context-Free Grammar

$$
\begin{array}{lclclclcl}
E & \rightarrow & N & \mid & E + E & \mid & E \times E & \mid & (E) \\
N & \rightarrow & 0 & \mid & 1 & \mid & N0 & \mid & N1
\end{array}
$$

Is a parse tree always unique? NO



Both parse trees yield the string $1 + 1 \times 1$.
This is called an ambiguity—discussed later in the course.

Exercise 1.3

Do the exercise 5.2.1 from the book.

Exercise 5.2.1: For the grammar and each of the strings in Exercise 5.1.2, give parse trees.

Exercise 5.1.2: The following grammar generates the language of regular expression $\mathbf{0^*1(0+1)^*}$:

$$\begin{array}{lcl} S & \rightarrow & A1B \\ A & \rightarrow & 0A \mid \epsilon \\ B & \rightarrow & 0B \mid 1B \mid \epsilon \end{array}$$

Give leftmost and rightmost derivations of the following strings:

* a) 00101.

b) 1001.

c) 00011.

Another Example

What is the language of the following grammar?

$$S \rightarrow S0S1 \mid S1S0 \mid \epsilon$$

Another Example

What is the language of the following grammar?

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: the set of strings with an **equal** number of 0's and 1's.

Let L_H be the set of all words over $T = \{0, 1\}$ that have an equal number of 0's and 1's.

Another Example

What is the language of the following grammar?

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: the set of strings with an **equal** number of 0's and 1's.

Let L_H be the set of all words over $T = \{0, 1\}$ that have an equal number of 0's and 1's.

Proof: We need to prove the hypothesis in two parts:

1. If $S \stackrel{\star}{\Rightarrow} w$ then $w \in L_H$ (the grammar only generates strings in L_H).
2. If $w \in L_H$ then $S \stackrel{\star}{\Rightarrow} w$ (all strings in L_H are generated by the grammar).

Structural Induction

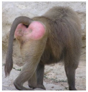
$$S \rightarrow S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \overset{\star}{\Rightarrow} w$, then $w \in L_H$

Structural Induction

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \stackrel{\star}{\Rightarrow} w$, then $w \in L_H$

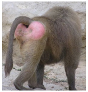


Proof idea: show that recursive inference never produces a word outside L_H.

Structural Induction

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \overset{\star}{\Rightarrow} w$, then $w \in L_H$



Proof idea: show that recursive inference never produces a word outside L_H.

This is called a **proof by structural induction**:

- prove the property holds in the base cases: rules with no variables in body.
- inductive cases: prove the property is preserved by all other rules

Structural Induction

$$S \rightarrow S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \overset{\star}{\Rightarrow} w$, then $w \in L_H$

Base case: $S \rightarrow \epsilon$.

Structural Induction

$$S \rightarrow S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \stackrel{\star}{\Rightarrow} w$, then $w \in L_H$

Base case: $S \rightarrow \epsilon$.
$\epsilon \in L_H$. ✓

Structural Induction

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \overset{\star}{\Rightarrow} w$, then $w \in L_H$

Base case: $S \to \epsilon$.
$\epsilon \in L_H$. ✓
Inductive step 1: $S \to S0S1$.

Structural Induction

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \overset{\star}{\Rightarrow} w$, then $w \in L_H$

Base case: $S \to \epsilon$.
$\epsilon \in L_H$. ✓

Inductive step 1: $S \to S0S1$.
If $w_1, w_2 \in L_H$, then also $w_1 0 w_2 1 \in L_H$. ✓

Structural Induction

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \overset{\star}{\Rightarrow} w$, then $w \in L_H$

Base case: $S \to \epsilon$.
$\epsilon \in L_H$. ✓
Inductive step 1: $S \to S0S1$.
If $w_1, w_2 \in L_H$, then also $w_1 0 w_2 1 \in L_H$. ✓
Inductive step 2: $S \to S1S0$.

Structural Induction

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $S \overset{\star}{\Rightarrow} w$, then $w \in L_H$

Base case: $S \to \epsilon$.
$\epsilon \in L_H$. ✓
Inductive step 1: $S \to S0S1$.
If $w_1, w_2 \in L_H$, then also $w_1 0 w_2 1 \in L_H$. ✓
Inductive step 2: $S \to S1S0$.
If $w_1, w_2 \in L_H$, then also $w_1 1 w_2 0 \in L_H$. ✓

Structural Induction

$$S \to S0S1 \mid S1S0 \mid \epsilon$$

Hypothesis: if $w \in L_H$ then $S \stackrel{\star}{\Rightarrow} w$

Proof idea: Use induction and possible "shapes" of words in L_H.
We know that w has the same number n of 0's and 1's (and nothing else).
We will use induction on n.

Base case: n is 0, i.e. $w = \epsilon$. The production $S \to \epsilon$ can be used to derive $S \stackrel{\star}{\Rightarrow} w$. ✓

Inductive step: $n > 0$.Then we know that w contains at least one 0 and one 1. We can distinguish two (symmetric) cases: either w ends with 0 or it ends with 1. Let us consider the first case, i.e. $w = w'0$. Now split w' into $w'_1 1 w'_2$ such that $w'_1 1$ is the shortest prefix of w' containing exactly one more 1 than 0's. Note that such a prefix necessarily exists. Clearly, w'_1 and w'_2 belong to L_H.
Since each of them contain less 0's and 1's than n we can use the inductive hypothesis to conclude that $S \stackrel{\star}{\Rightarrow} w'_1$ and $S \stackrel{\star}{\Rightarrow} w'_2$. We can then use production $S \to S1S0$ to conclude that $S \stackrel{\star}{\Rightarrow} w'_1 1 w'_2 0$.

Exercise Session

Complete exercises 1.1, 1.2 and 1.3 if you did not finish them.

Continue with the exercises in the next slides, where you will be asked to design context free grammars for some languages. In each exercise:

- provide an informal argument (a short paragraph) of why your grammar is correct (i.e. that it generates all and only the words in the language);
- provide 1-2 interesting examples of words, with their parse trees, and how they can be obtained by derivation or by inference.

Exercise 1.4 Grammars for non-regular languages

You may remember these languages from previous lectures...

Exercise 4.1.1 : ~~Prove that the following are not regular languages.~~

b) The set of strings of balanced parentheses. These are the strings of characters "(" and ")" that can appear in a well-formed arithmetic expression.

* c) $\{0^n10^n \mid n \geq 1\}$.

d) $\{0^n1^m2^n \mid n \text{ and } m \text{ are arbitrary integers}\}$.

e) $\{0^n1^m \mid n \leq m\}$.

f) $\{0^n1^{2n} \mid n \geq 1\}$.

Provide context-free grammars for each of the languages.
NOTE: n and m are non-negative integers.

Exercise 1.5 Grammars for lists

Let A be a set of elements. You can assume that elements in A are generated by a non-terminal A and you can consider a simple case for A (for example, the alphabet $\{0, 1\}$).

Design context a free grammar for lists of elements belonging to A.

Recall that lists can be inductively defined: a list of elements of A can be

- the empty list;
- or a single element of A followed by a list of elements of A;

Exercise 1.6 Grammars for trees

Let A be a set of elements. You can assume that elements in A are generated by a non-terminal A and you can consider a simple simple case for A (for example, the alphabet $\{0, 1\}$).

Design a context free grammar for binary trees with leafs belonging to A.

Recall that trees can be inductively defined: a tree with leafs from A can be

- the empty tree;
- or a node with with a left sub-tree with leafs from A and a right-tree with leafs from A;

Hint: start designing a way to represent a binary tree as a string. Use examples to drive the process.

1.7 Grammars for dictionaries

Let K be a set of keys and V be a set of values. A map (or dictionary) is a partial function $K \rightharpoonup V$, mapping keys into values. Consider the following grammar for maps:

$$\begin{array}{rcl} S & \rightarrow & \mathrm{nil} \mid K \mapsto V \mid S, S \\ K & \rightarrow & 0 \mid 1 \mid 2 \mid \cdots \mid n \\ V & \rightarrow & a \mid b \mid c \mid \cdots \mid z \end{array}$$

where nil denotes the empty map, K generates keys, and V generates values.

1. Does this grammar allow us to denote all possible maps from K to V?
2. Do all words admitted by the grammar denote meaningful maps?
3. Can a map be denoted with two different words?

Exercise 1.8 A grammar for regular expressions

Solve the following exercise from the book.

***! Exercise 5.1.5:** Let $T = \{0, 1, (,), +, *, \emptyset, e\}$. We may think of T as the set of symbols used by regular expressions over alphabet $\{0, 1\}$; the only difference is that we use e for symbol ϵ, to avoid potential confusion in what follows. Your task is to design a CFG with set of terminals T that generates exactly the regular expressions with alphabet $\{0, 1\}$.

Recall that the set of **regular expressions** (over alphabet A) can be inductively defined as follows:

- e and $\emptyset$ are regular expressions
- Every symbol of the alphabet A is a regular expression
- If E and F are already regular expressions, then also $E + F$ and EF and $E*$ and (E) are regular expressions.

where e is a syntactic symbol not belonging to A, used to denote the empty string.

Exercise 1.9 From regular expressions to context free grammars

Define a function that, given a regular expression E returns a grammar G that recognises exactly the same language as E.

Hint: define the function by induction on the structure of E.

What else?

Download and read the description of the mandatory assignment from CampusNet.