Automata Theory and Formal Languages

Summer Term 2026

3rd Lecture

Grammars

Prof. Dr. Matthias Mnich *Institute for Algorithms and Complexity*

Grammars - Motivating example

"The students study automata theory."

Grammars - Motivating example

"The students study automata theory."

Is the sentence grammatically correct,
that is, is it part of the English language?

Grammars - Motivating example

"The students study automata theory."

Is the sentence grammatically correct,
that is, is it part of the English language?

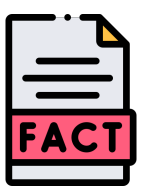
Formal condition: A *sentence* can be written in the form
nominal expression verb nominal expression

Grammars - Motivating example

"The students study automata theory."

Is the sentence grammatically correct,
that is, is it part of the English language?

Formal condition: A *sentence* can be written in the form
nominal expression verb nominal expression

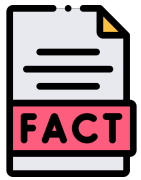


Observations.

- *article* **the** is followed by the *noun* **students**
- both together are a *nominal expression*
- **study** is a *verb*
- **automata theory** is also a nominal expression

Grammars - Motivating example

"The students study automata theory."

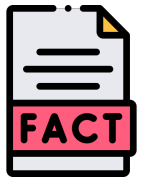


Observations.

- *article* **the** is followed by the *noun* **students**
- both together are a *nominal expression*
- **study** is a *verb*
- **automata theory** is also a nominal expression

Grammars - Motivating example

"The students study automata theory."



Observations.

- *article* **the** is followed by the *noun* **students**
- both together are a *nominal expression*
- **study** is a *verb*
- **automata theory** is also a nominal expression

sentence $\Rightarrow$ *nominal expression verb nominal expression*

$\Rightarrow$ *article noun verb nominal expression*

$\Rightarrow$ The students *verb nominal expression*

$\Rightarrow$ The students study *nominal expression*

$\Rightarrow$ The students study automata theory

Grammars - Formal introduction

Grammars are a formal tool to describe languages.

Grammars - Formal introduction

Grammars are a formal tool to describe languages.

The description of the grammar is always *finite*;
grammars allow to generate *infinite* languages.

Grammars - Formal introduction

Grammars are a formal tool to describe languages.

The description of the grammar is always *finite*;
grammars allow to generate *infinite* languages.

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

Grammars - Formal introduction

Grammars are a formal tool to describe languages.

The description of the grammar is always *finite*;
grammars allow to generate *infinite* languages.

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

- V: finite set of the *variables* or *non-terminal symbols*

Grammars - Formal introduction

Grammars are a formal tool to describe languages.

The description of the grammar is always *finite*;
grammars allow to generate *infinite* languages.

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

- V: finite set of the *variables* or *non-terminal symbols*
- Σ: terminal alphabet with $V \cap \Sigma = \emptyset$

Grammars - Formal introduction

Grammars are a formal tool to describe languages.

The description of the grammar is always *finite*;
grammars allow to generate *infinite* languages.

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

- V: finite set of the *variables* or *non-terminal symbols*
- Σ: terminal alphabet with $V \cap \Sigma = \emptyset$
- $S \in V$: start variable

Grammars - Formal introduction

Grammars are a formal tool to describe languages.

The description of the grammar is always *finite*;
grammars allow to generate *infinite* languages.

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

- V: finite set of the *variables* or *non-terminal symbols*
- Σ: terminal alphabet with $V \cap \Sigma = \emptyset$
- $S \in V$: start variable
- $\mathcal{P}$ is the set of *production rules* of the form
  left side $\rightarrow$ **right side**
  Formally: $\mathcal{P} \subset \underbrace{(V \cup \Sigma)^* \backslash \Sigma^*}_{\text{left side}} \times \underbrace{(V \cup \Sigma)^*}_{\text{right side}}$

Grammars

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

- V: finite set of the *variables* or *non-terminal symbols*
- Σ: terminal alphabet with $V \cap \Sigma = \emptyset$
- $S \in V$: start variable
- $\mathcal{P}$ is the set of *production rules* of the form
  left side $\rightarrow$ **right side**
  Formally: $\mathcal{P} \subset \underbrace{(V \cup \Sigma)^* \backslash \Sigma^*}_{\text{left side}} \times \underbrace{(V \cup \Sigma)^*}_{\text{right side}}$

Grammars

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

- V: finite set of the *variables* or *non-terminal symbols*
- Σ: terminal alphabet with $V \cap \Sigma = \emptyset$
- $S \in V$: start variable
- $\mathcal{P}$ is the set of *production rules* of the form
  left side $\rightarrow$ **right side**
  Formally: $\mathcal{P} \subset \underbrace{(V \cup \Sigma)^* \setminus \Sigma^*}_{\text{left side}} \times \underbrace{(V \cup \Sigma)^*}_{\text{right side}}$

Convention. Variables are usually capital letters, whereas terminal symbols are usually small letters.

Grammars

A *grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{P}, S)$, where

- V: finite set of the *variables* or *non-terminal symbols*
- Σ: terminal alphabet with $V \cap \Sigma = \emptyset$
- $S \in V$: start variable
- $\mathcal{P}$ is the set of *production rules* of the form
  left side $\rightarrow$ **right side**
  Formally: $\mathcal{P} \subset \underbrace{(V \cup \Sigma)^* \setminus \Sigma^*}_{\text{left side}} \times \underbrace{(V \cup \Sigma)^*}_{\text{right side}}$

Convention. Variables are usually capital letters, whereas terminal symbols are usually small letters.

$G = (\{S\}, \{a\}, \mathcal{P}, S)$ with $\mathcal{P} = \{S \rightarrow Sa, S \rightarrow a\}$

Generating languages from grammars

A grammar $G = (V, \Sigma, \mathcal{P}, S)$ *generates* a language $L(G)$ by applying the production rules from $\mathcal{P}$ successively, starting from the start variable S until no variables are present anymore.

Generating languages from grammars

A grammar $G = (V, \Sigma, \mathcal{P}, S)$ *generates* a language $L(G)$ by applying the production rules from $\mathcal{P}$ successively, starting from the start variable S until no variables are present anymore.

The derivation process.

Generating languages from grammars

A grammar $G = (V, \Sigma, \mathcal{P}, S)$ *generates* a language $L(G)$ by applying the production rules from $\mathcal{P}$ successively, starting from the start variable S until no variables are present anymore.

The derivation process.

A string $u \in (V \cup \Sigma)^*$ can be transformed into a string $v \in (V \cup \Sigma)^*$ if

$$u = xyz$$
$$v = xy'z \quad \text{with} \quad x, z \in (V \cup \Sigma)^*$$

where $y \to y'$ is a production rule in $\mathcal{P}$.

Generating languages from grammars

A grammar $G = (V, \Sigma, \mathcal{P}, S)$ *generates* a language $L(G)$ by applying the production rules from $\mathcal{P}$ successively, starting from the start variable S until no variables are present anymore.

The derivation process.

A string $u \in (V \cup \Sigma)^*$ can be transformed into a string $v \in (V \cup \Sigma)^*$ if

$$u = xyz$$
$$v = xy'z \quad \text{with} \quad x, z \in (V \cup \Sigma)^*$$

where $y \to y'$ is a production rule in $\mathcal{P}$.

Short: $u \underset{G}{\Rightarrow} v$ or $u \Rightarrow v$ if G can be determined from context.

Generating languages from grammars

For a derivation $\underset{G}{\Rightarrow}$, its *reflexive transitive closure* $\underset{G}{\Rightarrow}^*$ means multiple applications of $\underset{G}{\Rightarrow}$, in any combination and possibly repeated.

Generating languages from grammars

For a derivation $\underset{G}{\Rightarrow}$, its *reflexive transitive closure* $\underset{G}{\Rightarrow}^*$ means multiple applications of $\underset{G}{\Rightarrow}$, in any combination and possibly repeated.

For a grammar G, the language generated by G is given by

$$L(G) := \{w \in \Sigma^* \mid S \underset{G}{\Rightarrow}^* w\} \ .$$

Generating languages from grammars

$G = (\{S\}, \{a\}, \mathcal{P}, S)$ with $\mathcal{P} = \{S \to Sa, S \to a\}$.

Generating languages from grammars

$G = (\{S\}, \{a\}, \mathcal{P}, S)$ with $\mathcal{P} = \{S \to Sa, S \to a\}$.

Example derivation:

$$S \Rightarrow Sa \Rightarrow Saa \Rightarrow aaa \in \Sigma^*$$

Generating languages from grammars

$G = (\{S\}, \{a\}, \mathcal{P}, S)$ with $\mathcal{P} = \{S \to Sa, S \to a\}$.

Example derivation:

$$S \Rightarrow Sa \Rightarrow Saa \Rightarrow aaa \in \Sigma^*$$

Thus we have $aaa \in L(G)$.

Generating languages from grammars

$G = (\{S\}, \{a\}, \mathcal{P}, S)$ with $\mathcal{P} = \{S \to Sa, S \to a\}$.

Example derivation:

$$S \Rightarrow Sa \Rightarrow Saa \Rightarrow aaa \in \Sigma^*$$

Thus we have $aaa \in L(G)$.

The generated language is given by:

$$\begin{aligned} L(G) &= \{w \in \{a\}^* \mid S \underset{G}{\Rightarrow}^* w\} \\ &= \{w \in \{a\}^* \mid S \Rightarrow Sa \Rightarrow \cdots \Rightarrow \underbrace{a \ldots a}_{n \text{ times}}, n \in \mathbb{N}\} \\ &= \{a^n \mid n \in \mathbb{N}\} \end{aligned}$$

Generating languages from grammars

$$G = (\{E, T, F\}, \{a, +, *\}, \mathcal{P}, E)$$

$$\mathcal{P} = \{E \to T, E \to E + T, T \to F,$$
$$T \to T * F, F \to a\}$$

Generating languages from grammars

$$G = (\{E, T, F\}, \{a, +, *\}, \mathcal{P}, E)$$
$$\mathcal{P} = \{E \to T, E \to E + T, T \to F,$$
$$T \to T * F, F \to a\}$$

Then $a + a \in L(G)$ since:

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + F \Rightarrow a + a$$

Syntax trees

For grammars whose production rules $\mathcal{P}$ have only non-terminal symbols on the left-hand side, the derivation process can be visualized as a *syntax tree* (or *parse tree*).

Syntax trees

For grammars whose production rules $\mathcal{P}$ have only non-terminal symbols on the left-hand side, the derivation process can be visualized as a *syntax tree* (or *parse tree*).

For a grammar $G = (V, \Sigma, \mathcal{P}, S)$ and a derivation $S \Rightarrow^* \alpha$, the syntax tree is constructed as follows:

Syntax trees

For grammars whose production rules $\mathcal{P}$ have only non-terminal symbols on the left-hand side, the derivation process can be visualized as a *syntax tree* (or *parse tree*).

For a grammar $G = (V, \Sigma, \mathcal{P}, S)$ and a derivation $S \Rightarrow^* \alpha$, the syntax tree is constructed as follows:

- S is the root of the tree

Syntax trees

For grammars whose production rules $\mathcal{P}$ have only non-terminal symbols on the left-hand side, the derivation process can be visualized as a *syntax tree* (or *parse tree*).

For a grammar $G = (V, \Sigma, \mathcal{P}, S)$ and a derivation $S \Rightarrow^* \alpha$, the syntax tree is constructed as follows:

- S is the root of the tree
- If the rule $B \to X_1 X_2 \ldots X_k$ is applied:

Syntax trees

For grammars whose production rules $\mathcal{P}$ have only non-terminal symbols on the left-hand side, the derivation process can be visualized as a *syntax tree* (or *parse tree*).

For a grammar $G = (V, \Sigma, \mathcal{P}, S)$ and a derivation $S \Rightarrow^* \alpha$, the syntax tree is constructed as follows:

- S is the root of the tree
- If the rule $B \to X_1 X_2 \dots X_k$ is applied:
  - Add nodes $X_1, X_2, \dots X_k$ to tree.

Syntax trees

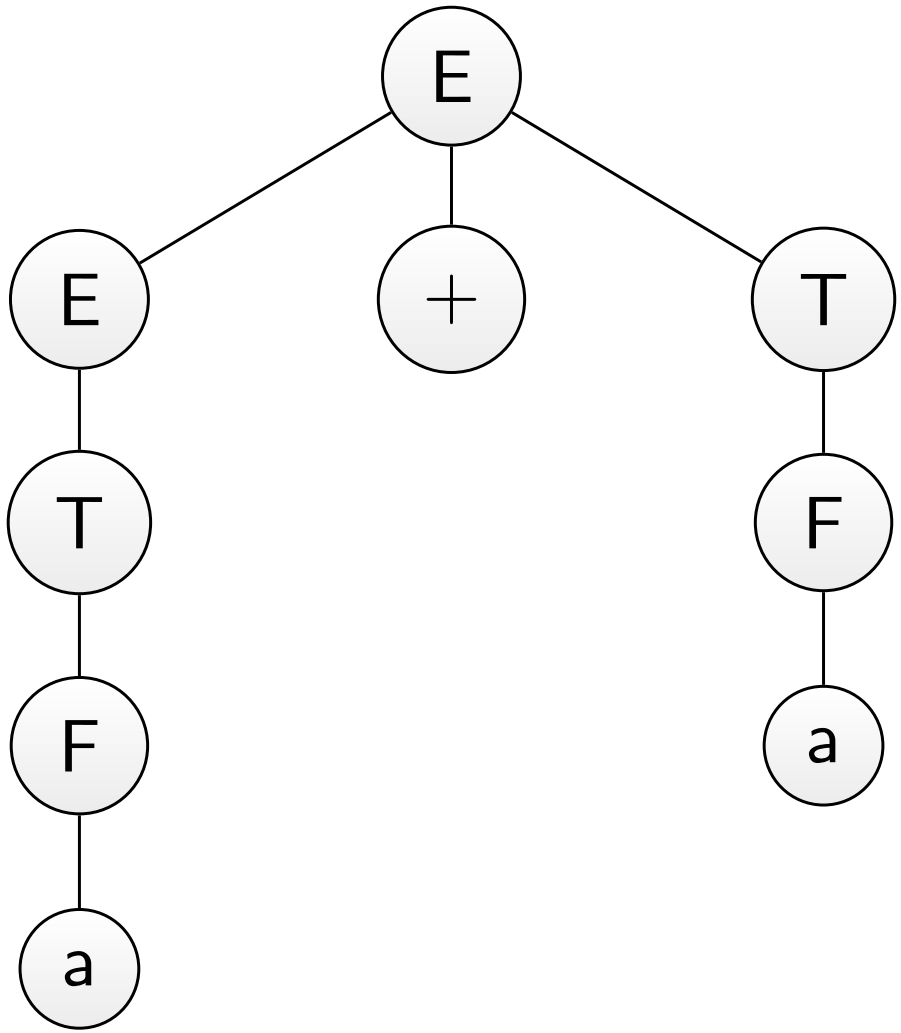
For grammars whose production rules $\mathcal{P}$ have only non-terminal symbols on the left-hand side, the derivation process can be visualized as a *syntax tree* (or *parse tree*).

For a grammar $G = (V, \Sigma, \mathcal{P}, S)$ and a derivation $S \Rightarrow^* \alpha$, the syntax tree is constructed as follows:

- S is the root of the tree
- If the rule $B \to X_1 X_2 \dots X_k$ is applied:
  - Add nodes $X_1, X_2, \dots X_k$ to tree.
  - connect $B - X_1, B - X_2, \dots, B - X_k$ with an edge

An example of a syntax tree

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + F \Rightarrow a + a$$



Properties of syntax trees

- In the syntax tree, one cannot see in which order the production rules have been applied to the string.

Properties of syntax trees

- In the syntax tree, one cannot see in which order the production rules have been applied to the string.
- Derivation is not necessarily a deterministic, but generally a *non-deterministic* process.

Properties of syntax trees

- In the syntax tree, one cannot see in which order the production rules have been applied to the string.
- Derivation is not necessarily a deterministic, but generally a *non-deterministic* process.
- A derivation is only complete if the final string w contains only terminal symbols, i.e. $w \in \Sigma^*$.