TDDD14 / TDDD85 – Lecture 7

Context-free Grammars

August Ernstsson, 2024 (based on lecture notes by Jonas Wallgren)

LINKÖPING UNIVERSITY

About me

- Postdoc at IDA

- Research and interests
  - High-level parallel programming languages, concepts, libraries
  - Heterogeneous computer architectures (multi-core and GPU programming)
  - High-performance computing (clusters, supercomputers)

- Languages, parsers, syntax trees etc. are *valuable tools* in my own research

Coming up next in the course

- <u>This week</u>
  - **Today**: Context-free grammars (CFG) introduction
  - **Wednesday**: CFG rewriting, GFG normal forms

- <u>Next week</u>
  - **Monday**: Pushdown automata (PDA)
  - **Friday**: Equivalence between CFG and PDA

- Then three weeks with one lecture per week
  - Properties of CFGs and parsing methods for CFGs

Let’s start!

Introduction

- The language $\{ 0^n1^n \mid n \geq 0\}$ is not regular
  - We were unable to handle it with the formalisms so far
- In this part of the course, we will introduce those that allow us to!
- We will start with notation, and get to automata next week.

Context-free Grammars

<expression> ::= <expression> * <expression>
| <expression> + <expression>
| <number>
<number> ::= <digit> <number>
| <digit>
<digit> ::= 0|1|2|3|4|5|6|7|8|9

- BNF: Backus-Naur Form

Example 1

E → E*E | E+E | N

N → DN | D

D → 0|1|2|3|4|5|6|7|8|9

- A grammar for expressions (E) of numbers (N) of digits (D)
- Each line is called a *production* (sometimes *rule* is used)
- ”→” can be read ”is composed of”
  - Different syntax is used, e.g. ::= or ←
  - Often < > are used to denote nonterminals (not part of the actual string)

Abbreviations for combining productions

<expression> ::= <expression> * <expression>
| <expression> + <expression>
| <number>

Equivalent to

<expression> ::= <expression> * <expression>
<expression> ::= <expression> + <expression>
<expression> ::= <number>

Abbreviations for combining productions

<expression> ::= <expression> * <expression>
| <expression> + <expression>
| <number>

Equivalent to

<expression> ::= <expression> * <expression>
<expression> ::= <expression> + <expression>
<expression> ::= <number>

And to

<expression> ::= <expression> * <expression> | <expression> + <expression> | <number>

Definition 1: Context-free grammars

- A *grammar* is a quadruple $G = \langle N, \Sigma, P, S \rangle$ where

  N = set of *nonterminals*

  Σ = set of *terminals* (the alphabet)

  $P \subseteq N \times (N \cup \Sigma) *$ = set of *production rules*

  $S \in N$ = *start symbol*

- P is a set of elements with
  - A left-hand side that is a nonterminal
  - A right-hand side that is a mix of terminals and nonterminals

Derivations

- Performing a derivation
  - Begin with the start symbol S;
  - step by step, use the production rules in P;
  - finally, end up with a string of nonterminals.
- In general, we use small greek letters for sequences of nonterminals and terminals.
  - **Example**: $\alpha, \beta, \gamma \in (N \cup \Sigma)*$
- Capital latin letters stand for nonterminals.

Derivations

- Performing a derivation
  - Begin with the start symbol S;
  - step by step, use the production rules in P;
  - finally, end up with a string of nonterminals.
- In general, we use small greek letters for sequences of nonterminals and terminals.
  - **Example**: $\alpha, \beta, \gamma \in (N \cup \Sigma)*$
- Capital latin letters stand for nonterminals.
- So, **if** we have reached $\alpha A \gamma$ **and** in the grammar there is a rule $A \rightarrow \beta$ **then** we can get $\alpha\beta\gamma$
  - i.e. the middle A has been replaced by the β from the right-hand-side of the grammar rule.
  - This is written $\alpha A \gamma \Rightarrow \alpha\beta\gamma$

Derivations, cont.

- $\alpha A\gamma \Rightarrow \alpha\beta\gamma$ is one *context-free* derivation step. We don't care about what α and γ are.
- We can *always* do the replacement of A with β. We can *ignore the context* of A.

Derivations, cont.

- $\alpha A\gamma \Rightarrow \alpha\beta\gamma$ is one *context-free* derivation step. We don't care about what α and γ are.
- We can *always* do the replacement of A with β. We can *ignore the context* of A.

- Several derivation steps one after another is denoted by $\Rightarrow*$
  - **Example:** $E \Rightarrow E+E \Rightarrow N+E \Rightarrow N+N \Rightarrow DN+N \Rightarrow 1N+N \Rightarrow \cdots \Rightarrow 123+456$
    - Thus $E \Rightarrow* 123+456$

$$E \rightarrow E{*}E \mid E{+}E \mid N$$
$$N \rightarrow DN \mid D$$
$$D \rightarrow 0|1|2|3|4|5|6|7|8|9$$

Definition 2: Context-free languages

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

- Language L of grammar G is
  - the set of all strings of terminals from the alphabet Σ
  - that can be derived from the start symbol S in zero or more steps.

- The language of a CGF is called a *context-free language* (CFL).
- The strings are of finite length, but the grammar is generally infinite.

Example 2

- Grammar G1:
  $N = \{ X \}$
  $\Sigma = \{ a, b \}$
  $S = X$
  $P = \{ X \to aXb \mid \varepsilon \}$

$X \Rightarrow \varepsilon$, $X \Rightarrow^* \varepsilon$

$X \Rightarrow aXb \Rightarrow ab$, $X \Rightarrow^* ab$

$X \Rightarrow aXb \Rightarrow aaXbb \Rightarrow aabb$, $X \Rightarrow^* aabb$

- Thus $\{ \varepsilon, ab, aabb \} \subseteq L(G1)$

Example 3

- Grammar G2:

  $N = \{ X \}$

  $\Sigma = \{ a, b \}$

  $S = X$

  $P = \{ X \rightarrow aXa \mid bXb \mid a \mid b \mid \varepsilon \}$

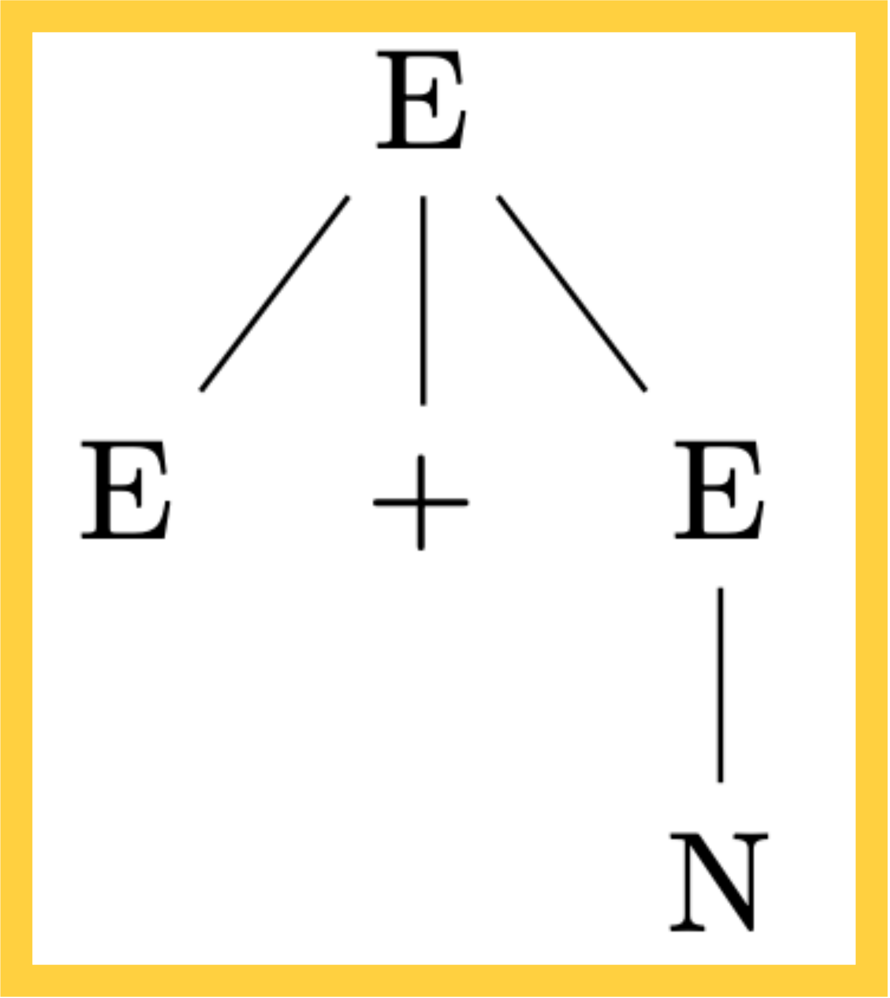
$X \Rightarrow aXa \Rightarrow abXba \Rightarrow abbXbba \Rightarrow abbbba$

$X \Rightarrow bXb \Rightarrow baXab \Rightarrow babab$

- It seems like $L(G2) = \{ x \in \{a, b\}^* \mid x = \text{reverse}(x) \}$, i.e. *palindromes* over Σ.

Derivation trees (parse trees)

- The derivation $E \Rightarrow E{+}E \Rightarrow E{+}N \Rightarrow \cdots$ could be depicted in a tree.
- Start symbol (here E) is found in the root.



- The tree shows *where* productions were applied
  - but not in which step-*order*.
- ”Derivation tree” is the more theoretical term.
- ”Parse tree” is more often used in practical contexts, e.g. parsing programs.

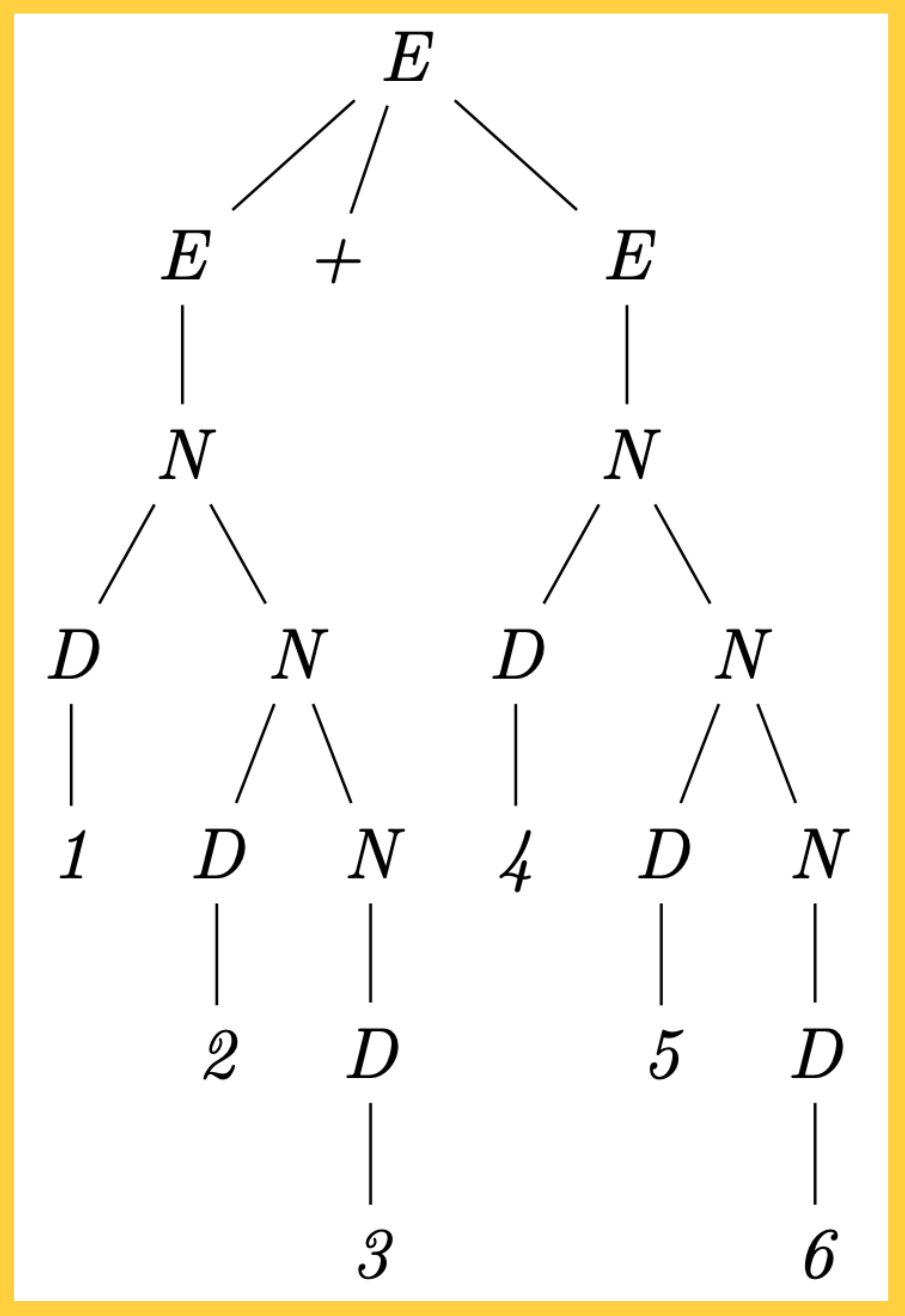
Definition 3: Derivation trees

- A *derivation tree* is a tree such that:
  - The *root* of a derivation tree is S.
  - Each *leaf* of a derivation tree $\in \Sigma$.
  - Each *inner node* of a derivation tree $\in$ N.
  - **If** the node A has the children p, q, r, …
    - **then** there is a rule A $\rightarrow$ pqr… $\in$ P

Example 4

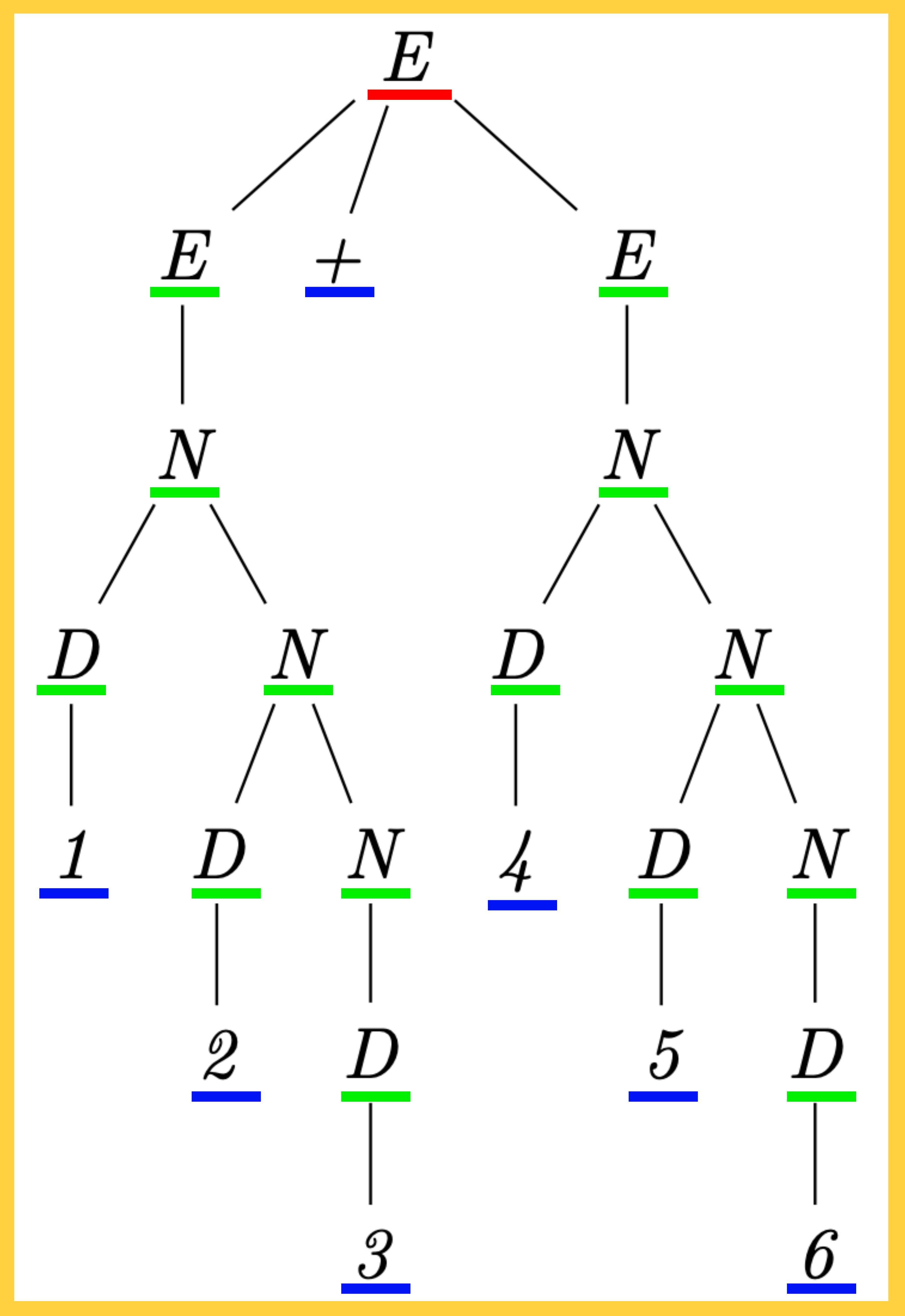
- The derivation tree for the string **123+456**:

E → E*E | E+E | N
N → DN | D
D → 0|1|2|3|4|5|6|7|8|9



Example 4, cont.

- A *derivation tree* is a tree such that:
  - The *root* of a derivation tree is S.
  - Each *leaf* of a derivation tree ∈ Σ.
  - Each *inner node* of a derivation tree ∈ N.
  - **If** the node A has the children p, q, r, …
    - **then** there is a rule A → pqr… ∈ P



Left and right (-most) derivations

- We will consider the grammar $E \rightarrow E*E \mid E+E \mid a \mid b \mid c$

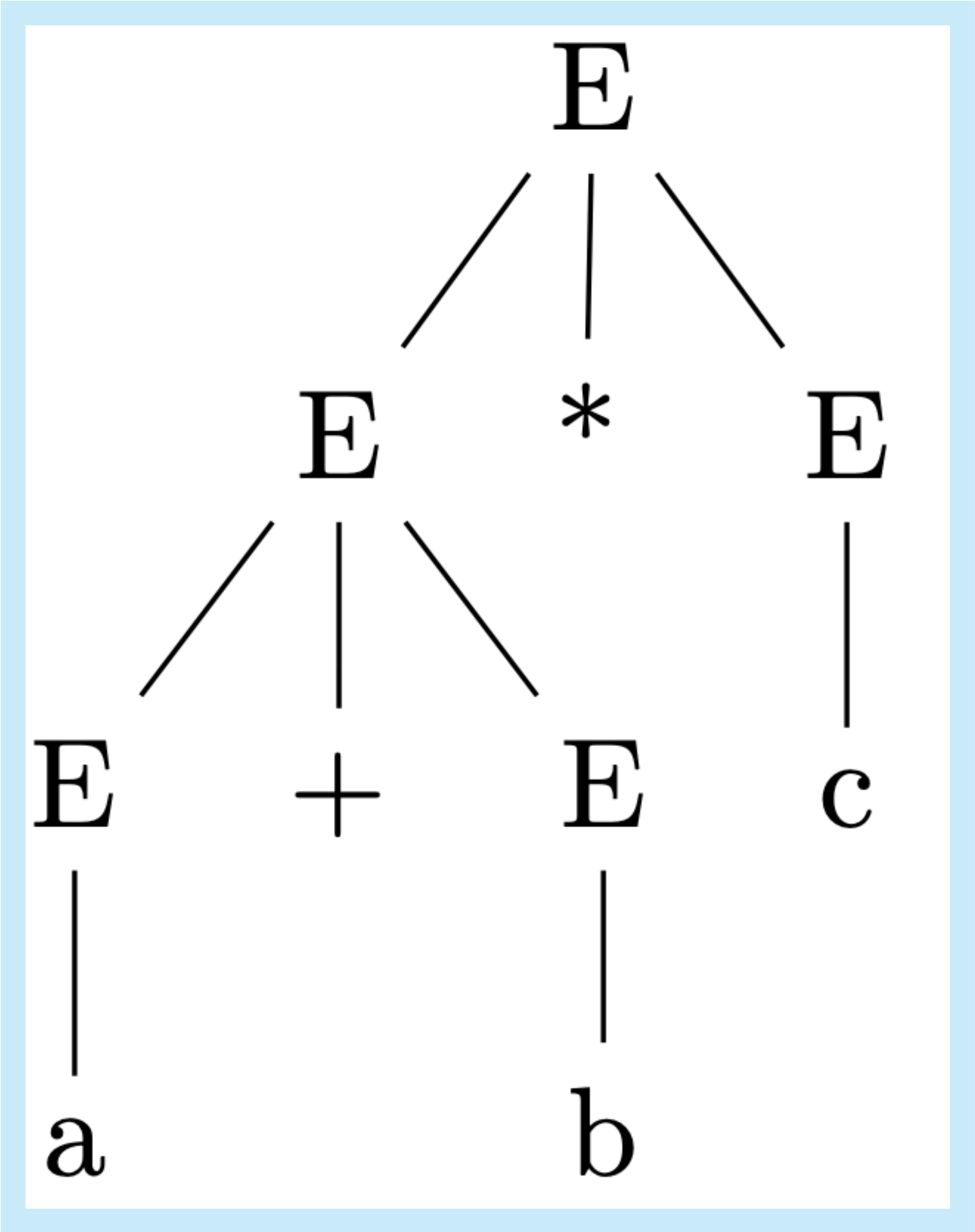
Left and right (-most) derivations

- We will consider the grammar $E \rightarrow E*E \mid E+E \mid a \mid b \mid c$
- <u>An example derivation</u>
  - $\underline{E} \Rightarrow \underline{E}*E \Rightarrow \underline{E}+E*E \Rightarrow a+\underline{E}*E \Rightarrow a+b*\underline{E} \Rightarrow a+b*c$
  - In each step, the *leftmost* nonterminal has been chosen
  - Called a *leftmost derivation*, symbol: $\Rightarrow_{lm}$

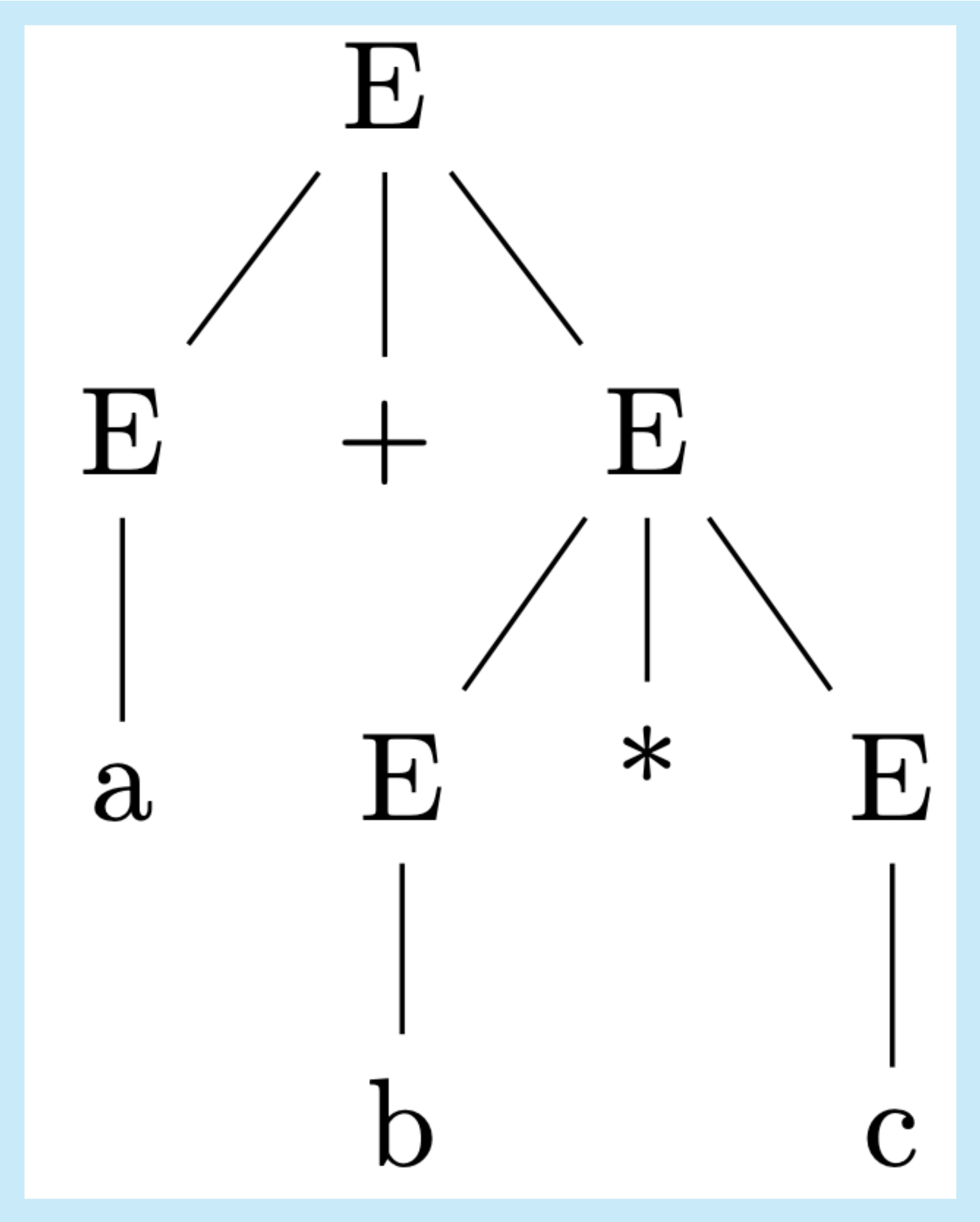
Left and right (-most) derivations

- We will consider the grammar $E \rightarrow E{*}E \mid E{+}E \mid a \mid b \mid c$
- <u>An example derivation</u>
  - $\underline{E} \Rightarrow \underline{E}{*}E \Rightarrow \underline{E}{+}E{*}E \Rightarrow a{+}\underline{E}{*}E \Rightarrow a{+}b{*}\underline{E} \Rightarrow a{+}b{*}c$
  - In each step, the *leftmost* nonterminal has been chosen
  - Called a *leftmost derivation*, symbol: $\Rightarrow_{lm}$
- <u>Another derivation</u>
  - $\underline{E} \Rightarrow E{+}\underline{E} \Rightarrow E{+}E{*}\underline{E} \Rightarrow E{+}\underline{E}{*}c \Rightarrow \underline{E}{+}b{*}c \Rightarrow a{+}b{*}c$
  - In each step, the *rightmost* nonterminal has been chosen
  - Called a *rightmost derivation*, symbol: $\Rightarrow_{rm}$

Leftmost and rightmost derivation trees

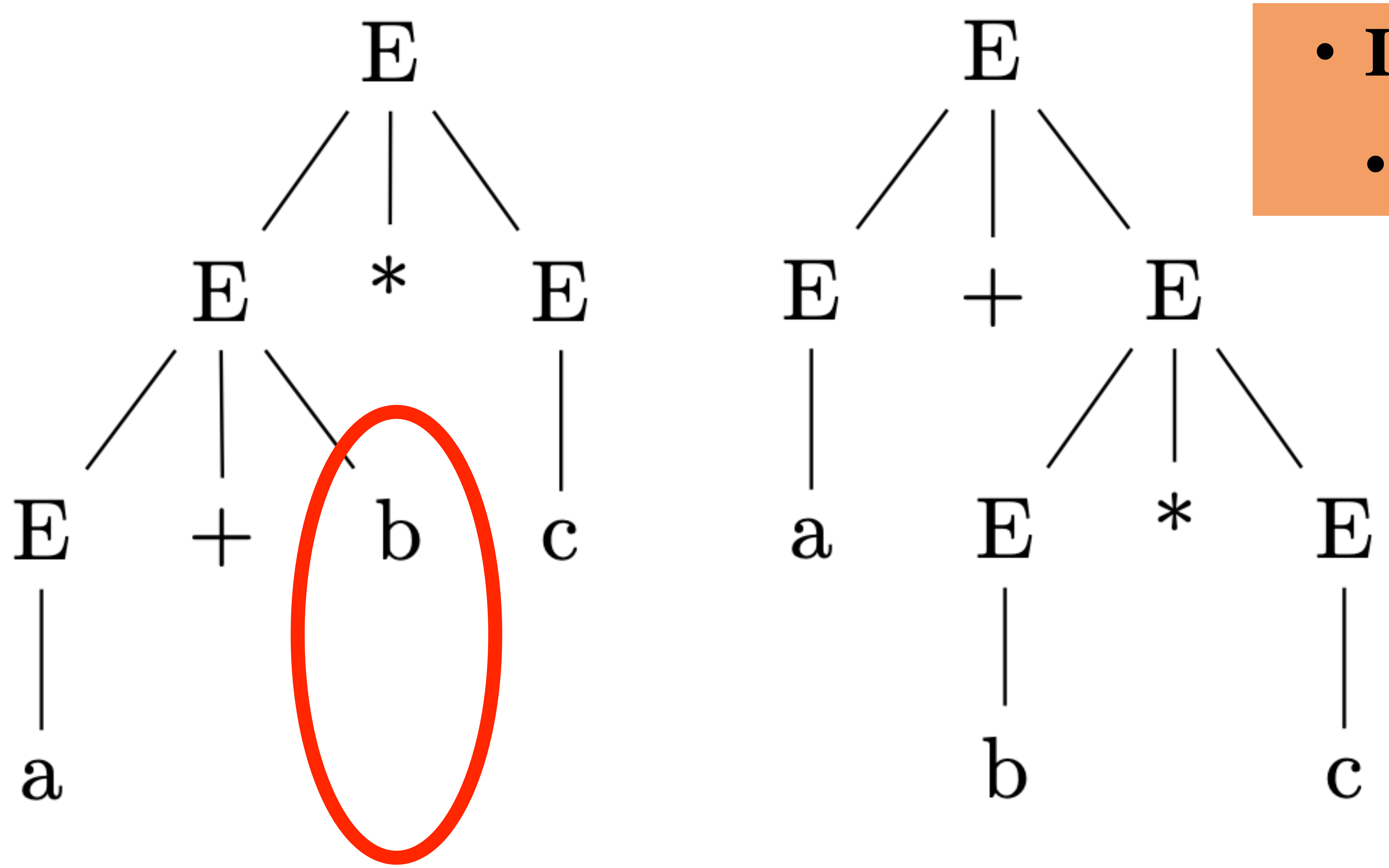


Leftmost



Rightmost

Oops! Error in published lecture notes?



- **If** the node A has the children p, q, r, …
  - **then** there is a rule A → pqr… ∈ P

E → E*E | E+E | a | b | c

Ambiguities

- We would like to analyze every string in exactly one way
- Different derivation/parse trees indicate an ambiguous grammar

- The grammar $E \to E*E \mid E+E \mid a \mid b \mid c$ handles arithmetic expressions
- If we evaluate a+b*c by means of the parse tree, we will get different results
  - Addition first
  - Multiplication first

- Solution? Rewrite the grammar

Example 5

- Unambiguous arithmetic expression grammar

$$E \rightarrow E+T \mid E-T \mid T$$
$$T \rightarrow T*F \mid T/F \mid F$$
$$F \rightarrow (E) \mid a \mid b \mid c$$

- Different nonterminals for expressions (E), terms (T), and factors (F)
- Standard *priorities* and *associativities*
- We also have bracketed expressions with the expected priority

Final note

- Goes back to lecture 1 …

- All regular languages are context free
  - Set of regular languages are a subset of all context-free languages

- Building up the different classes of formal langauges and their relations!

To think about

- Is your favorite programming / markup language context-free?
- Is there a CFG for $\{ a^n b^n c^n \mid n \geq 0 \}$?

Bonus: C++ standards document, annex A

Not part of the course material!

Annex A (informative)
Grammar summary [gram]

1 This summary of C++ syntax is intended to be an aid to comprehension. It is not an exact statement of the language. In particular, the grammar described here accepts a superset of valid C++ constructs. Disambiguation rules (6.8, 7.1, 10.2) must be applied to distinguish expressions from declarations. Further, access control, ambiguity, and type rules must be used to weed out syntactically valid but meaningless constructs.

Thanks for today!

LINKÖPING UNIVERSITY