Compiling Techniques
ECOTE - p.4
Grammars and introduction to parsing

DSc eng. Ilona Bluemke

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund

G = < T, N, P, S >

T - set of terminals

N **-** set of nonterminals

P – set of productions

$$P \in (T \cup N)^* N (T \cup N)^* \times (T \cup N)^*$$

$<\mu, \delta> \in P$ could be written as $\mu \rightarrow \delta$,

$$\mu \in (T \cup N)^* N (T \cup N)^*$$

$$\delta \in (T \cup N)^*$$

S – starting symbol, $S \in N$

Notational conventions:

1. These symbols are usually **nonterminals**:
- lower case names such as expression, statement, operator, etc.
- capital letters in the beginning of the alphabet
- letter **S**, when it appears, is usually the **start symbol**

2. These symbols are usually **terminals**:
- single lower case letters a,b,c,..., digits
- operator symbols such as +, -, etc.
- punctuation symbols such as parentheses, coma
- boldface strings such as **id**, **if**

3. Capital letters near the end of the alphabet: X,Y,Z represent sequences of nonterminals (nonterminal words)
4. Small letters near the end of the alphabet: x,y,z represent sequences of terminals (terminal words)
5. Lower case Greek letters α, β - nonterminal or terminal symbols
6. Lower case Greek letters μ, δ - nonterminal or terminal words (sequences of symbols)
7. If $A \rightarrow \alpha_1$, $A \rightarrow \alpha_2$, $A \rightarrow \alpha_3$, (A productions) may be written as $A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3$

Derivation

- $\mu \Rightarrow \delta$ — μ derives δ in **one step**

$\mu=\alpha A\beta \quad \wedge \quad \delta=\alpha\gamma\beta \ \wedge \ A \rightarrow \gamma \in P$

- $\mu \Rightarrow^k \delta$ — μ derives δ in **k steps**

$\exists (\mu_1, \mu_2, \ldots \mu_k) \quad \mu_1=\mu \quad \wedge \quad \mu_k= \delta \quad \wedge$

$\forall (2<=i<=k: \mu_{i-1} \Rightarrow \mu_i)$

- $\mu \Rightarrow^* \delta$ — μ derives δ in **zero or more steps**
- $\mu \Rightarrow^+ \delta$ — μ derives δ in **one or more steps**

Language generated by grammar *G*

$$L(G) = \{x \in T^* : S \Rightarrow^* x\}$$

Example 1

$G = < \{0,1\}, \{A, S\},$

$\{S \rightarrow 0A1, A \rightarrow 0A1, A \rightarrow \varepsilon\}, S >$

$S \Rightarrow 0A1$

$\Rightarrow 00A11$

$\Rightarrow 000A111$

$\Rightarrow 000111$

$L(G) = \{0^n1^n : n>0\}$

Example 2

$G_0 = <\{a, +, *, (,)\}, \{A, B, S\}, P, S>$

$P = \{S \rightarrow S+A \mid A,\ A \rightarrow A*B \mid B,\ B \rightarrow (S) \mid a\}$

$S \Rightarrow S+A \Rightarrow A+A \Rightarrow B+A \Rightarrow a+A \Rightarrow a+A*B \Rightarrow$
$a+B*B \Rightarrow a+a*B \Rightarrow a+a*a$

Left derivation – left nonterminal expanded

$S \Rightarrow^{+} a + a * a$

Example 3

$G = < \{a, b, c\}, \{A, B, S\}, P, S >$

$P = \{S \rightarrow aSAB \mid abB,\ BA \rightarrow AB,\ bA \rightarrow bb,\ bB \rightarrow bc,\ cB \rightarrow cc\ \}$

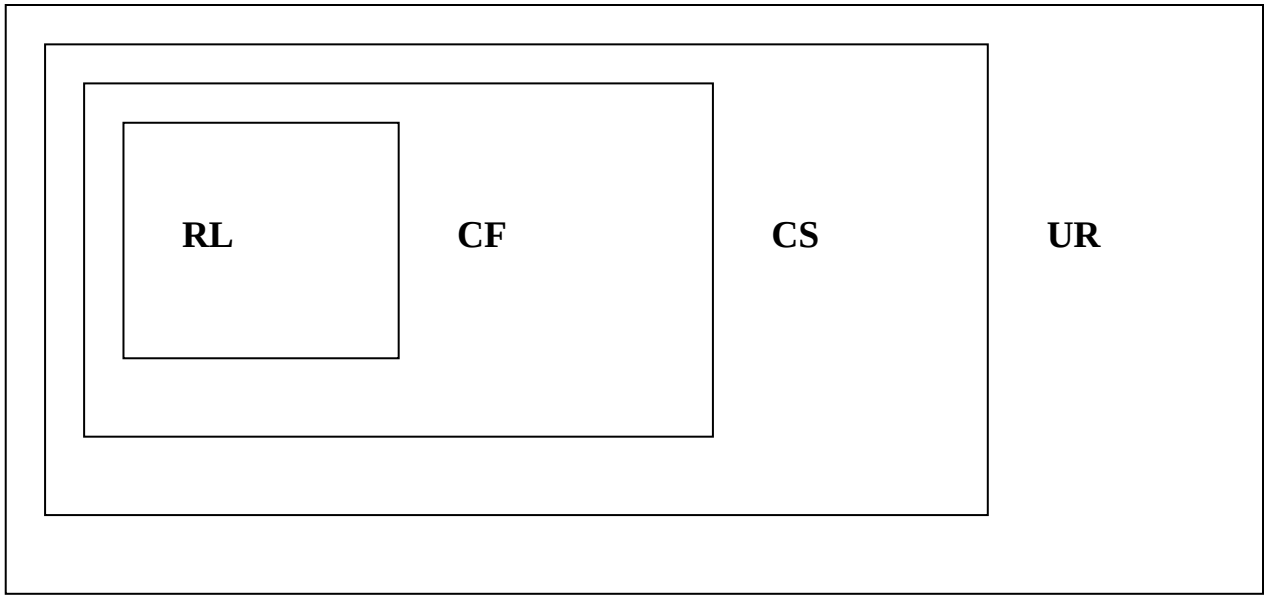
$S \Rightarrow aSAB \Rightarrow aabBAB \Rightarrow aabABB \Rightarrow aabbBB \Rightarrow aabbcB \Rightarrow aabbcc$

$L(G) = \{a^n b^n c^n : n >= 1\}$

Chomsky grammars classification

- **Type 3 (RL) right linear**
  G is right linear $\Leftrightarrow P \subset N \times (T^* \cup T^* N)$
  Productions: $A \rightarrow x$ or $A \rightarrow xB$
- **Type 2 (CF) context free**
  G is context free $\Leftrightarrow P \subset N \times (T \cup N)^*$
  Productions: $A \rightarrow \mu$
- **Type 1 (CS) context sensitive**
  G is context sensitive $\Leftrightarrow \forall\ (\varphi \rightarrow \psi \in P)$
  $\varphi = \mu A \delta \Rightarrow \psi = \mu \rho \delta \wedge \rho \in (T \cup N)^+$
  or $\forall\ (\varphi \rightarrow \psi \in P\ |\varphi| <= |\psi|$
  ($|\psi|$ the length of word ψ)
- **Type 0 (UR) unrestricted**

Chomsky hierarchy



Chomsky grammars classification

Type	Grammar	Production	Recognition
0	Recursive enumerable	No restriction	Turing machine
1	Context sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$ Albo $\alpha \rightarrow \beta$, $\lvert\alpha\rvert \leq \lvert\beta\rvert$	Automata linearly limited
2	Context free	$A \rightarrow \alpha$	Push down automata
3	Regular	$A \rightarrow a$ $A \rightarrow aB$ albo $A \rightarrow Ba$	Finite Automata

Parse trees and derivations

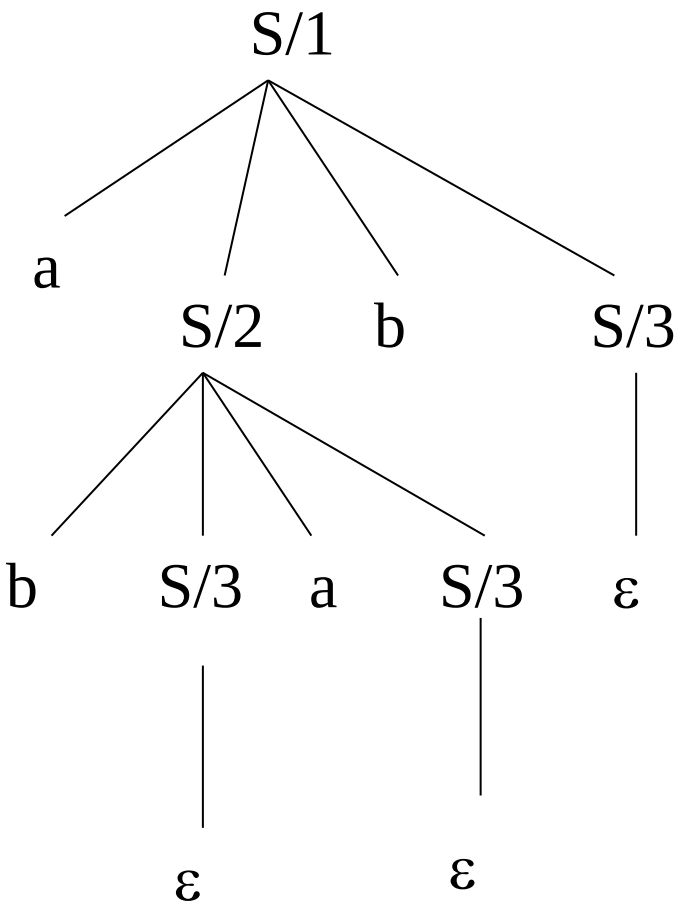
$G = <\{a, b\}, \{S\}, P, S>$

$P = \{1.\ S \rightarrow aSbS,\ 2.\ S \rightarrow bSaS,\ 3.\ S \rightarrow \varepsilon\ \}$

parse tree

graphical representation for derivations

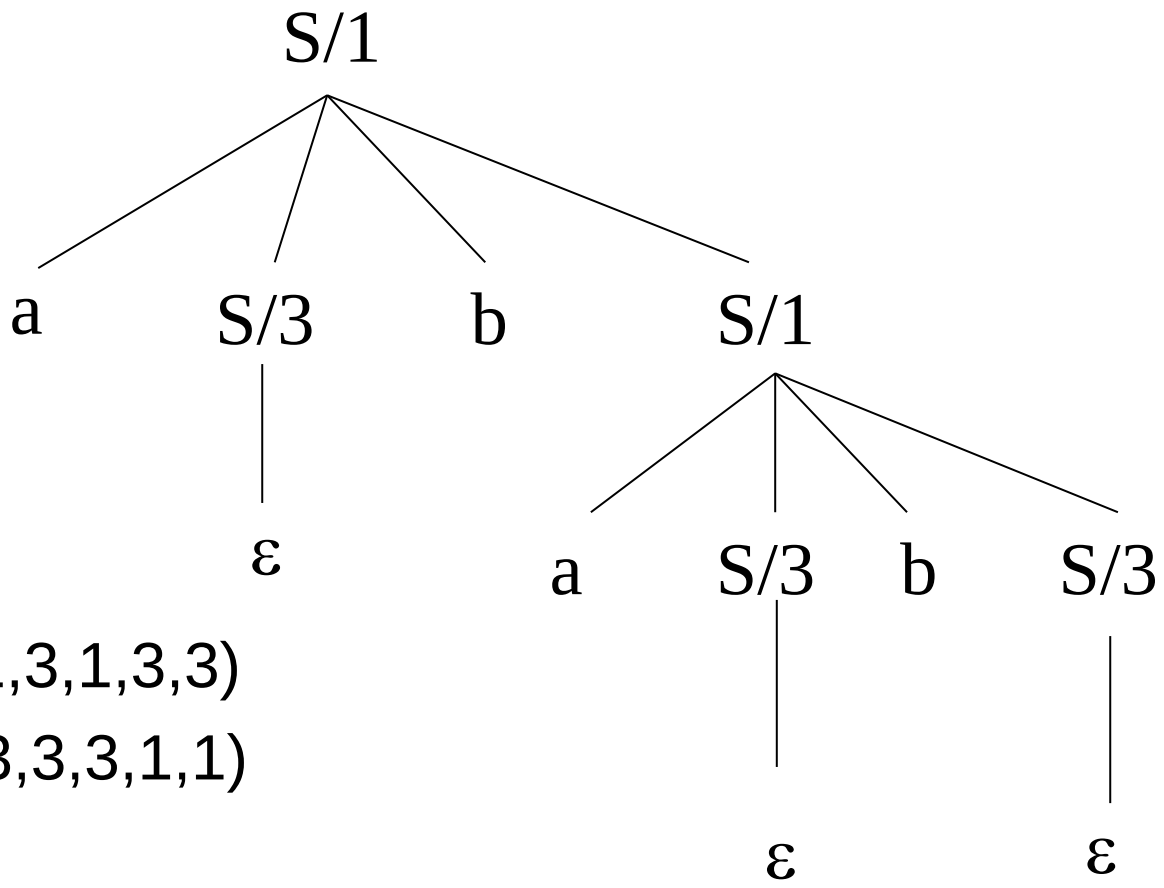
P = {1. S→aSbS, 2. S→bSaS, 3. S →ε }



lder(abab)=
(1,2,3,3,3)

rder(abab)=
(3,3,2,3,1)

P = {1. S→aSbS, 2. S→bSaS, 3. S →ε }



lder(abab)= (1,3,1,3,3)

rder(abab)= (3,3,3,1,1)

Leafs (terminals) read from left to right constitute a sentence, called **yield** or **frontier** of the tree.

Parse tree ignores variations in the order in which symbols are replaced.

Every parse tree has associated with it a **unique leftmost and rightmost derivation.**

- **leftmost** derivation – leftmost nonterminal is replaced at every step (preorder traversal)

- **rightmost** derivation – rightmost nonterminal is replaced at every step

A grammar that produces more than one parse tree for some sentence is said to be **ambiguous.**

Parsers

Parser for a grammar *G* produces a parse tree for sentence ***w*** or an error message if ***w*** is not a sentence of ***G*.**

Basics types of parsers for CF grammars:

- **top – down** (start with the root and work down to the leafs, recursive descent)
- **bottom – up** (build tree from the leafs to the root, shift-reduce)

Input is scanned from left to right, one symbol at a time.

Top-down parsing (with backtracking) for grammars without left recursion

- an attempt to find a **leftmost** derivation for input string
- starts from the root (S)
- initially **active node** is the starting symbol

- Left recursion (in production) eg. A->Aa

Top-Down

1. **if** the active node is a **nonterminal**, **then** use the first production for this symbol
2. active node becomes the leftmost node
3. **if** the active node is a **terminal** **then** goto 4 **else** goto 1
4. **if** current input symbol is the same as the symbol at active node **then** advance the input pointer; set new active node (next from the left); goto 1 **else** start **backtracking**
5. **if** the active node is **ε** **then**; set new active node (next from the left)

Backtracking in T-D

a) Remove production (one level of a parse tree), reset input pointer (if terminals are in removed production); change active node
b) Use alternative, next production for nonterminal if exists, otherwise goto a).
c) If all alternatives were check and the frontier of the tree is not the input sequence then the input sequence is not generated by the grammar

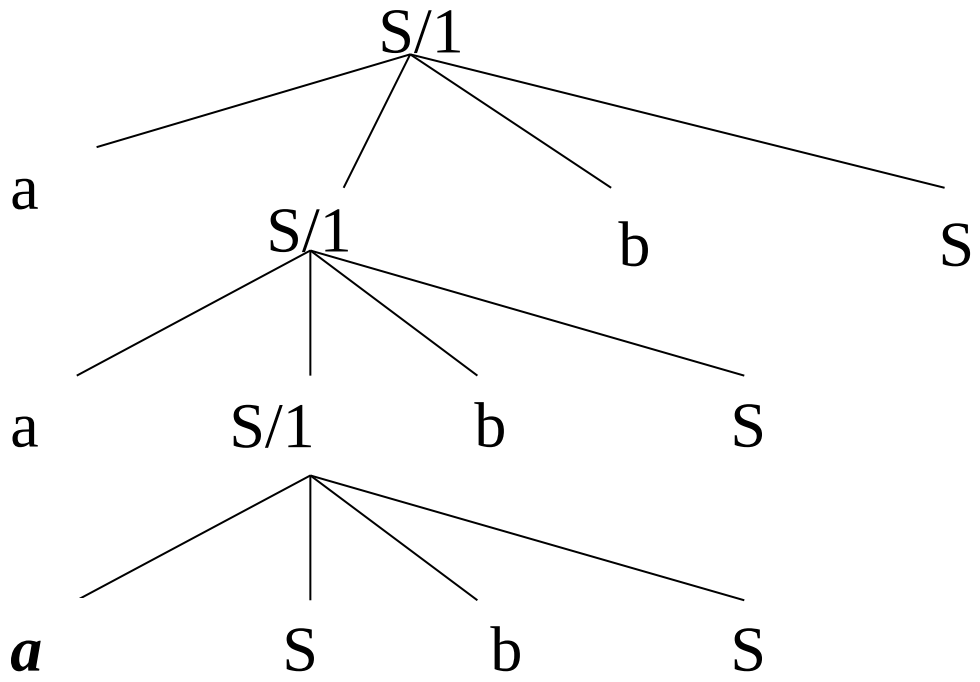
Example

$G = <\{a, b, c\}, \{S\}, P, S>$

$P = \{1.\ S \rightarrow aSbS,\ 2.\ S \rightarrow aS,\ 3.\ S \rightarrow c\ \}$

w = aacbc$

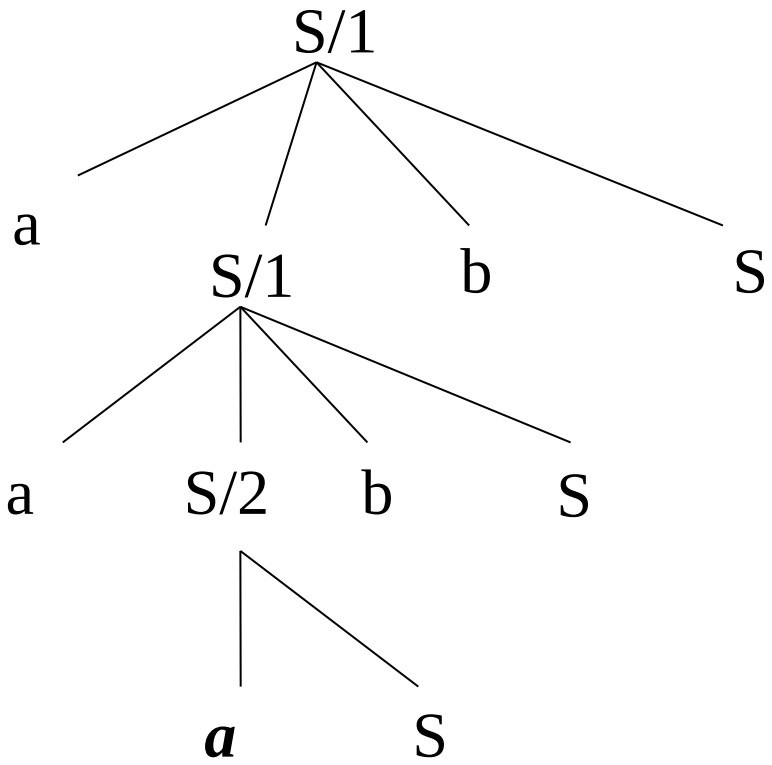
Part 1- aacbc$



aacbc$

P = {1. S→aSbS, 2. S→aS, 3. S →c }

Part 2- aacbc$

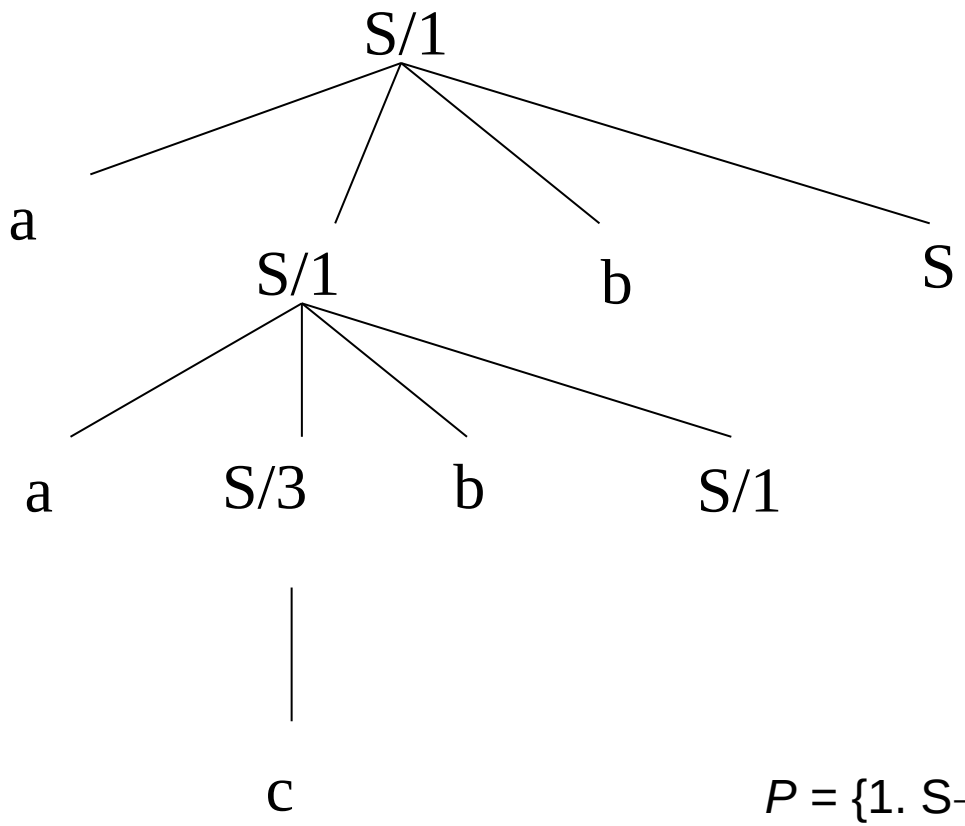


aacbc$

↑

P = {1. S→aSbS, 2. S→aS, 3. S →c }

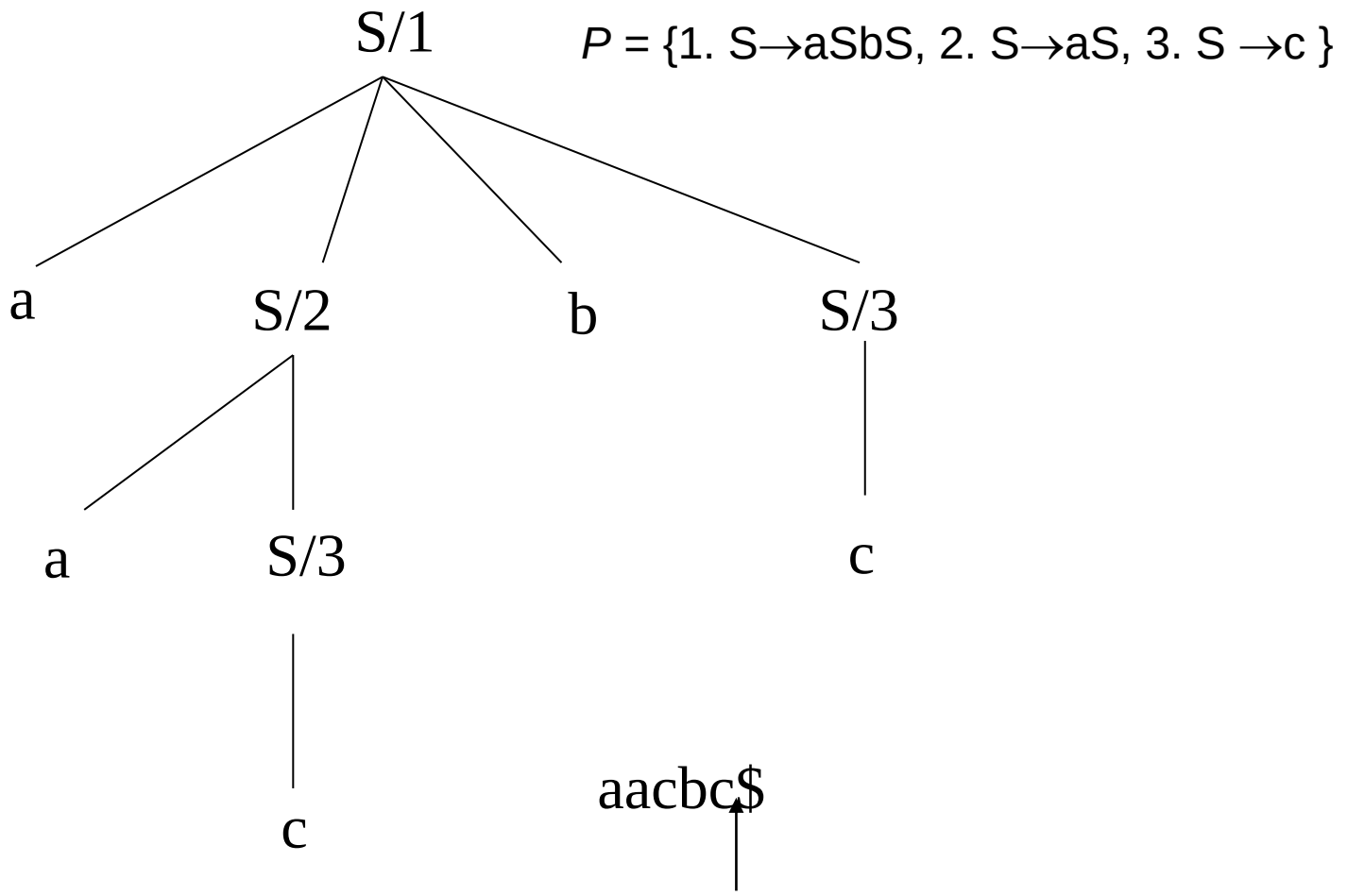
Part 3- aacbc$



P = {1. S→aSbS, 2. S→aS, 3. S →c }

aacbc$

Derivation tree



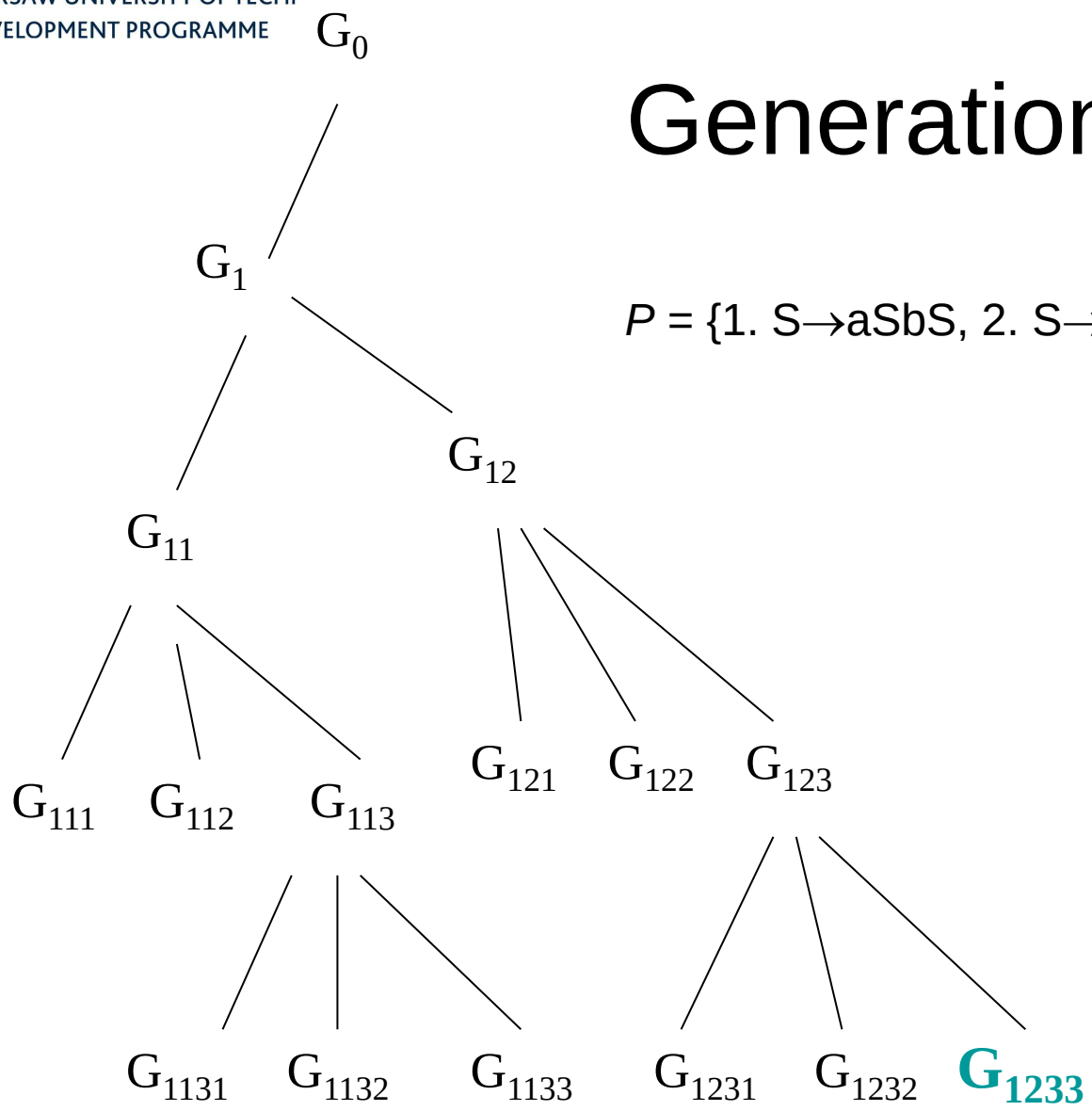
Left derivations

- lder(w) = {1,2,3,3}
- lder(w) = {2,1,3,3}

P = {1. S$\rightarrow$aSbS, 2. S$\rightarrow$aS, 3. S $\rightarrow$c }

Generation tree

P = {1. S→aSbS, 2. S→aS, 3. S →c }



BOTTOM-UP parsing (with backtracking)

- grammars **without cycles and ε-productions**
- shift – reducing
- an attempt to find in reverse a **rightmost** derivation for input string
- starts from the leafs (bottom) and works up towards the root
- Cycle: A =>* A, ε-production : A-> ε

1.

at each reduction step a particular substring matching the right side of a production (first matching) is replaced by the symbol on the left of that production

at each step all possible reductions are made

2.

if reductions are not possible **and** still there are input symbols **then** shift (advance) the input pointer; goto 1

3. **if** all input symbols were read **and** parse tree not obtained **then**

backtracking

- Remove production (one level of a parse tree), reset input pointer (if terminals in removed production)
- Use alternative reductions
- **if** all alternatives were check **and** the tree is not obtained **and** input pointer is at the first symbol **then** the input sequence is not generated by the grammar

Example

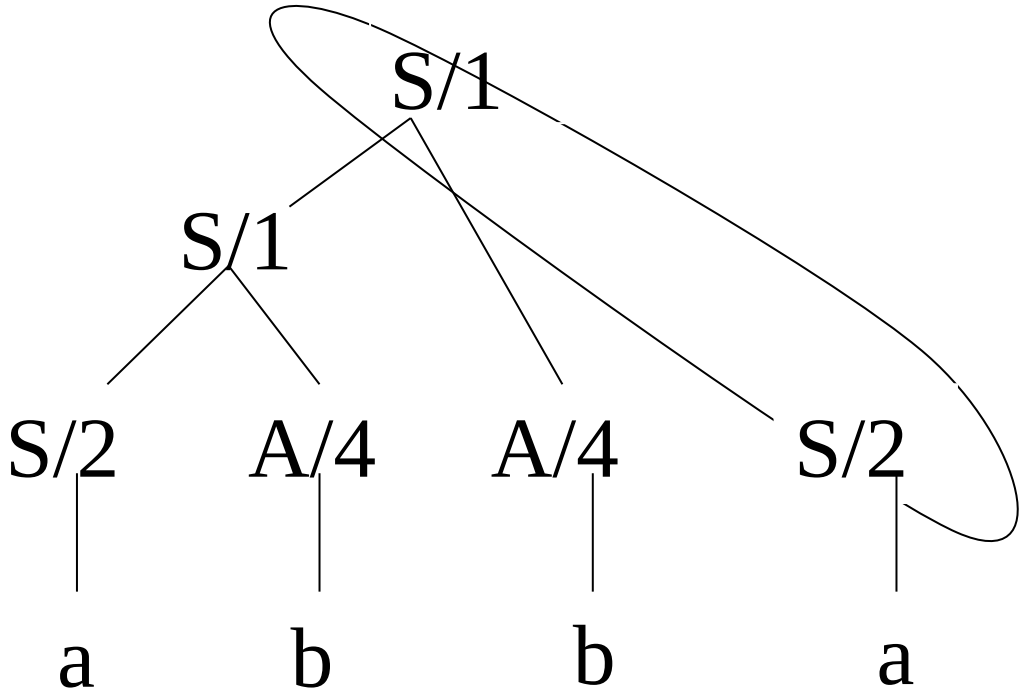
$G = <\{a, b\}, \{A, S\}, P, S>$

$P = \{1.\ S \rightarrow SA,\ 2.\ S \rightarrow a,\ 3.\ A \rightarrow AbS,$
$4.\ A \rightarrow b\ \}$

w = abba$

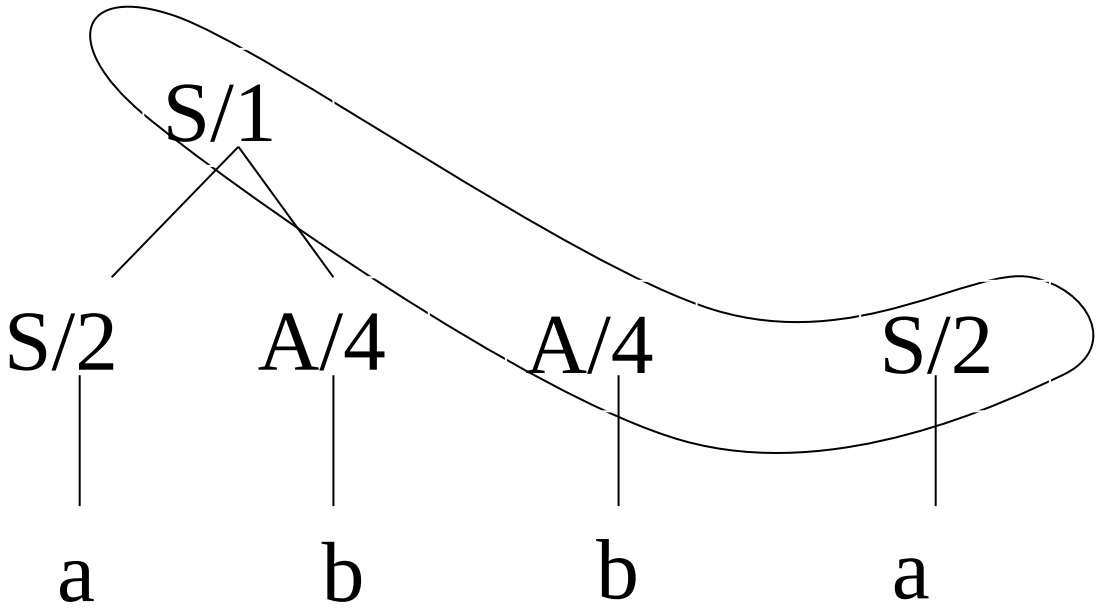
1.

P = {1. S→SA, 2. S→a, 3. A →AbS, 4. A→b }



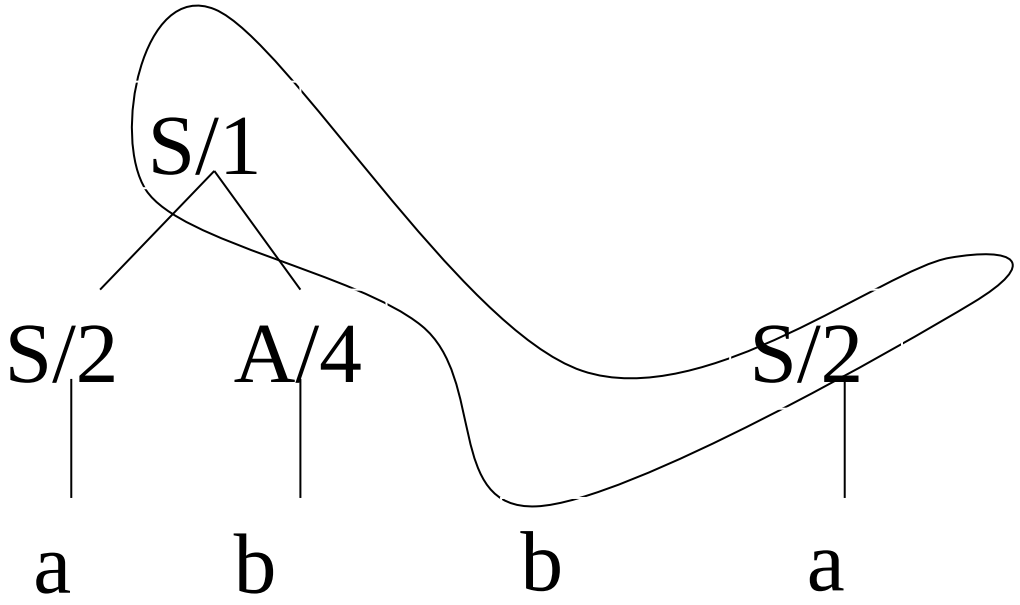
2.

P = {1. S→SA, 2. S→a, 3. A →AbS, 4. A→b }

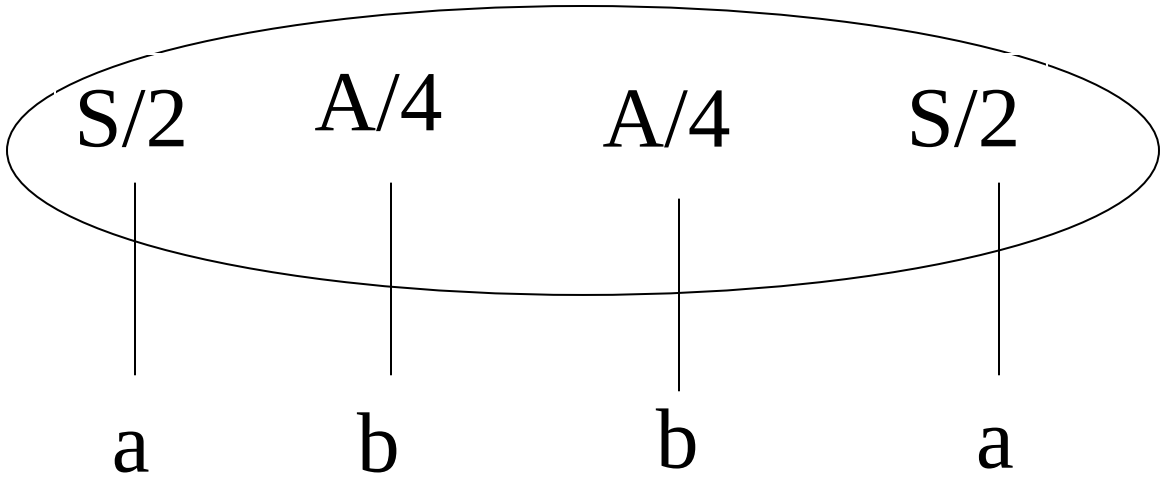


3.

P = {1. S→SA, 2. S→a, 3. A →AbS, 4. A→b }



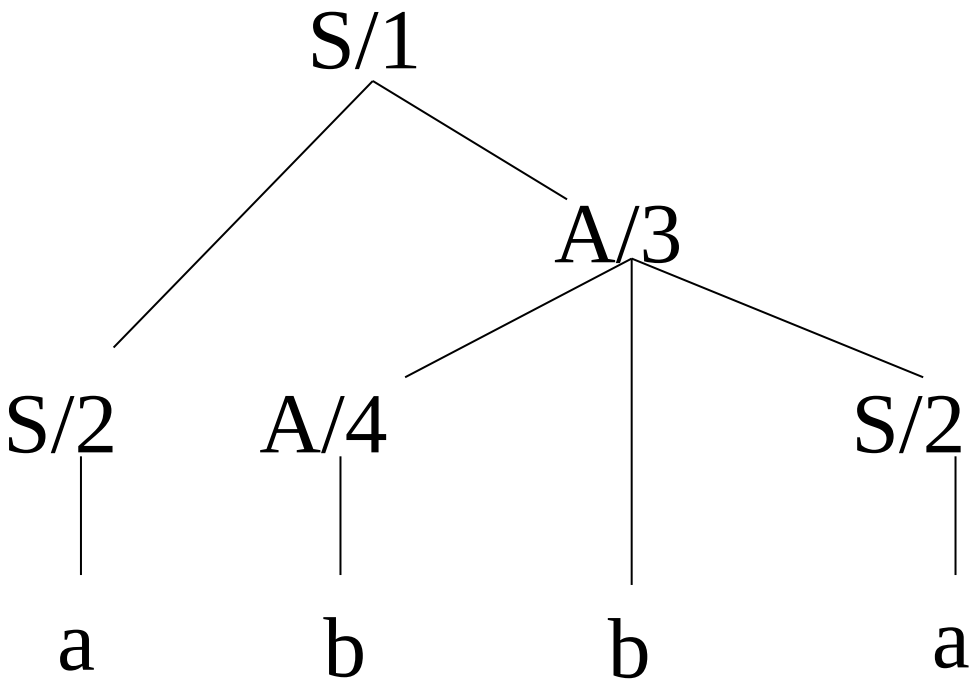
4.



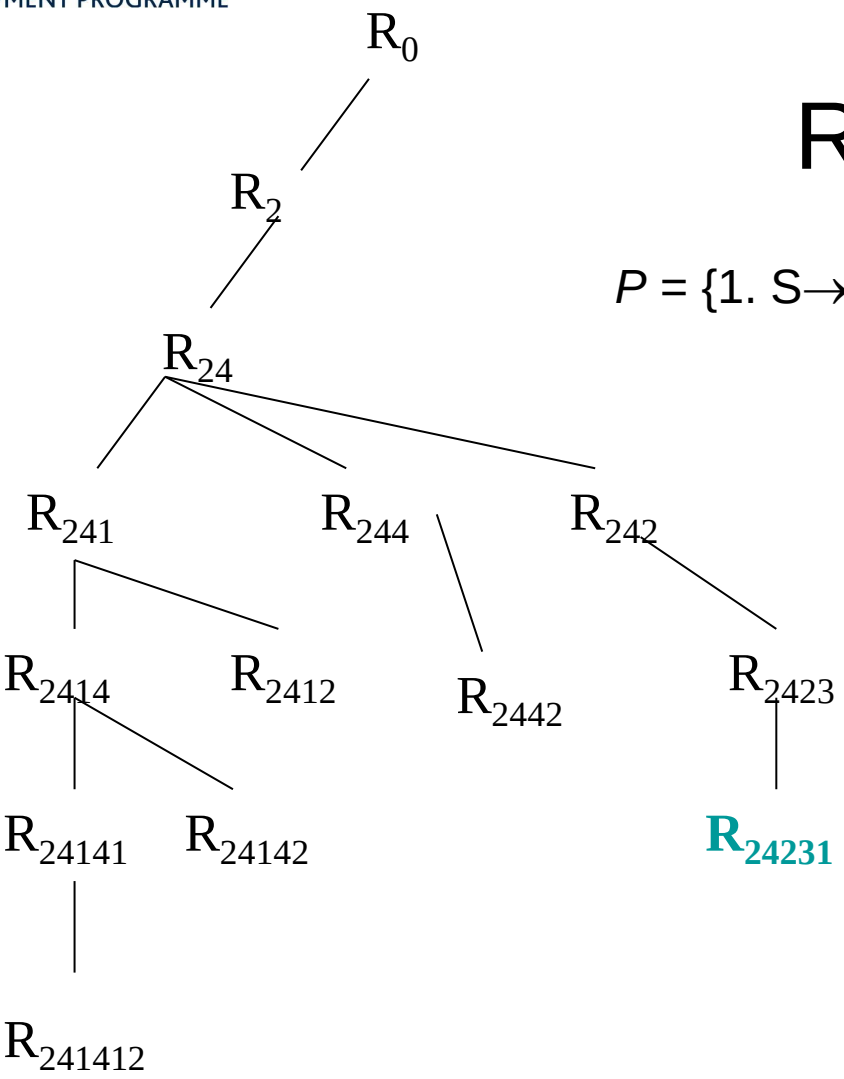
P = {1. S→SA, 2. S→a, 3. A →AbS, 4. A→b }

5.

$P = \{1.\ S \rightarrow SA,\ 2.\ S \rightarrow a,\ 3.\ A \rightarrow AbS,\ 4.\ A \rightarrow b\ \}$



Reduction tree



P = {1. S→SA, 2. S→a, 3. A →AbS, 4. A→b }

rder(w) = (2,4,2,3,1)

Grammars and introduction to parsing
end of part 4

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund