

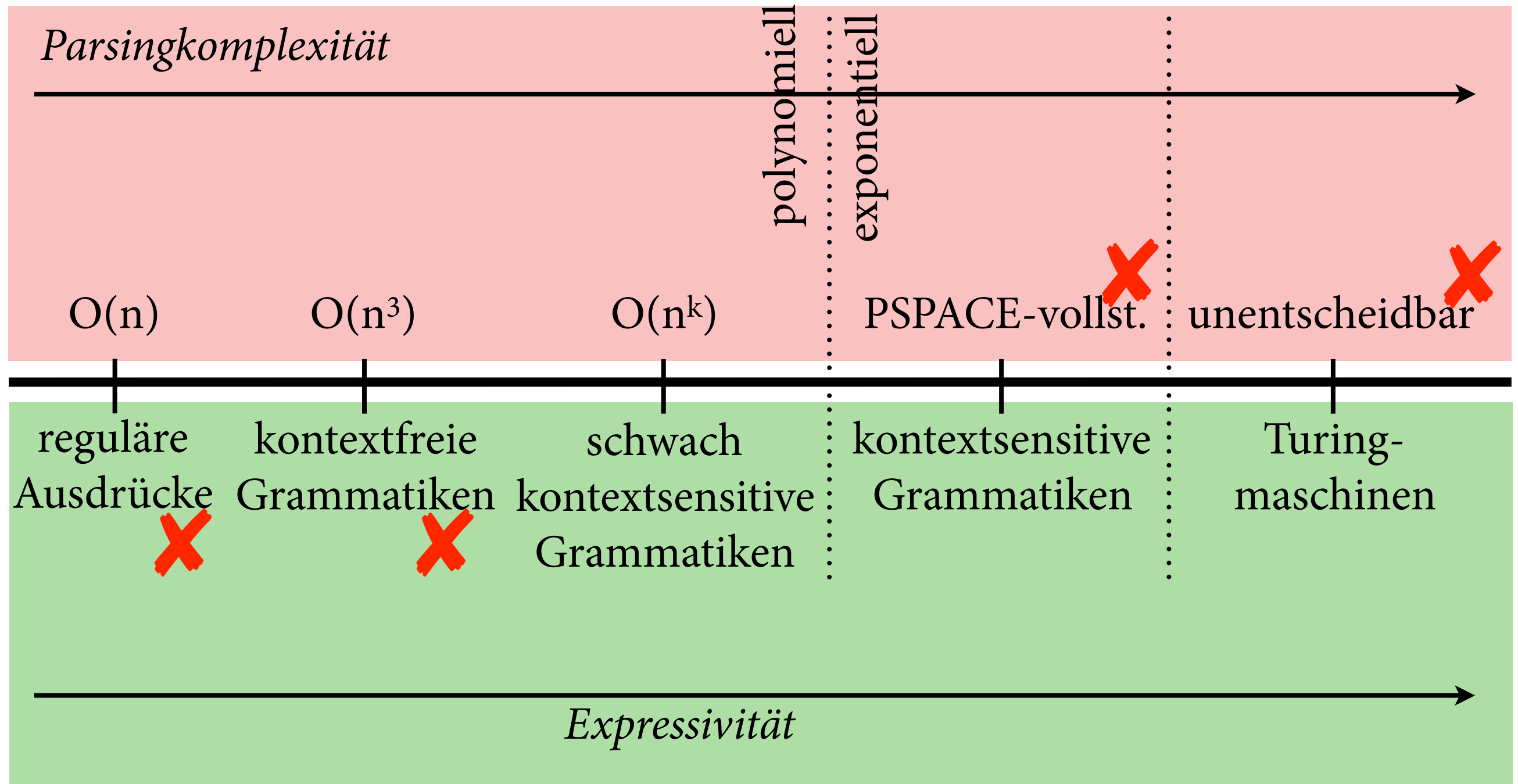
# **TAG:**

# **Parsing und Expressivität**

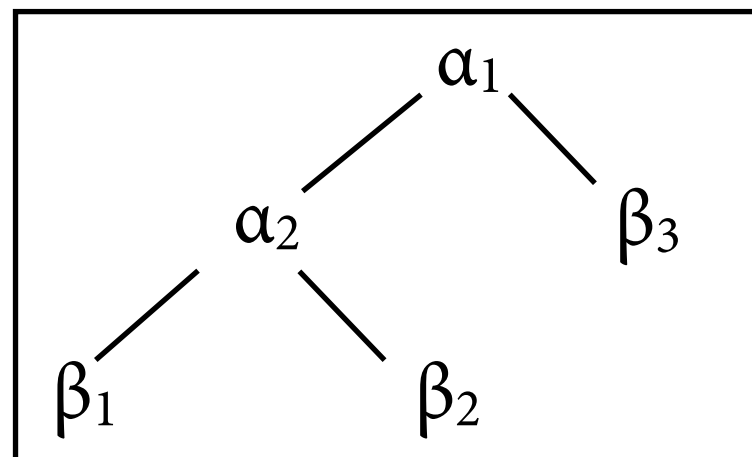
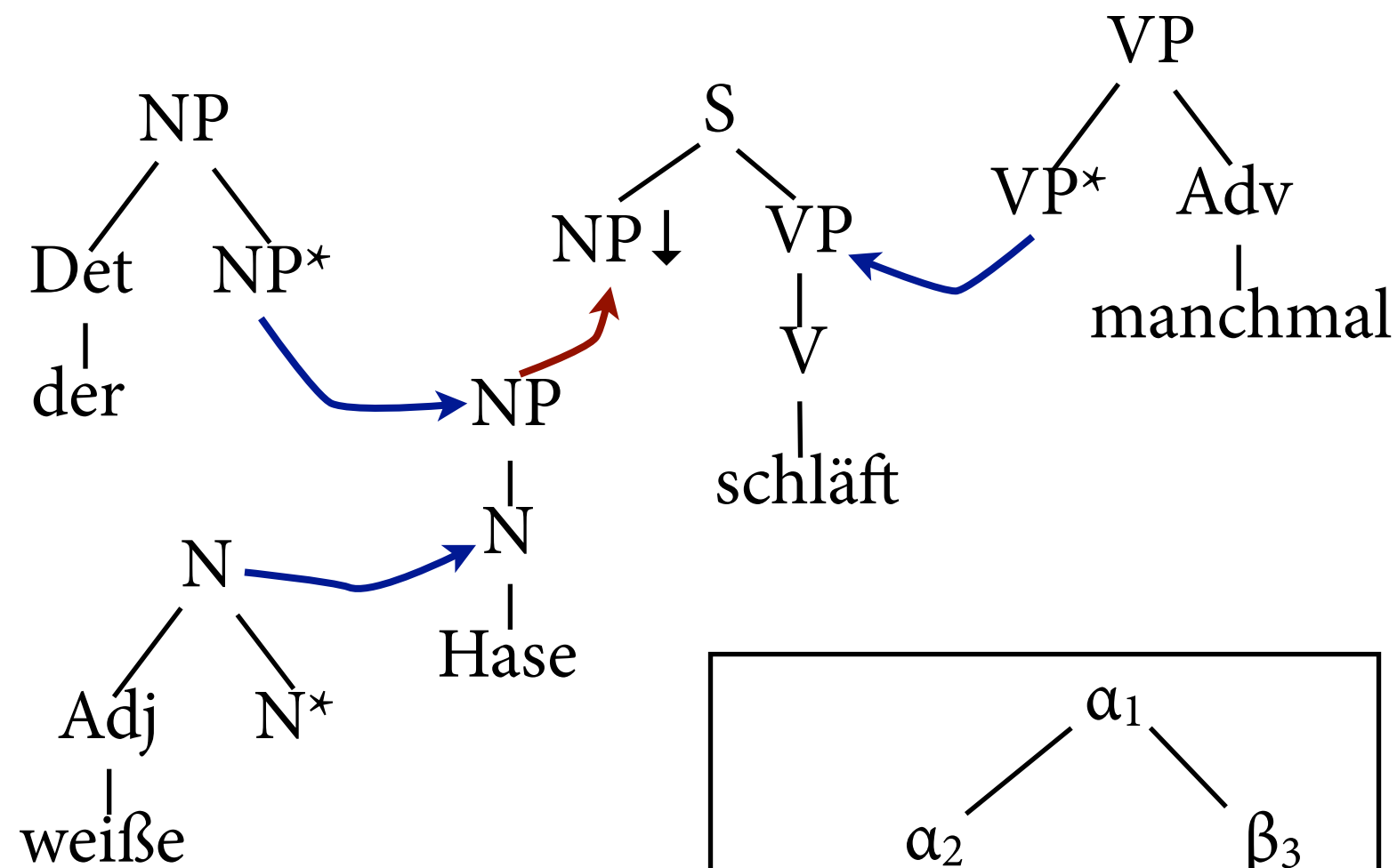
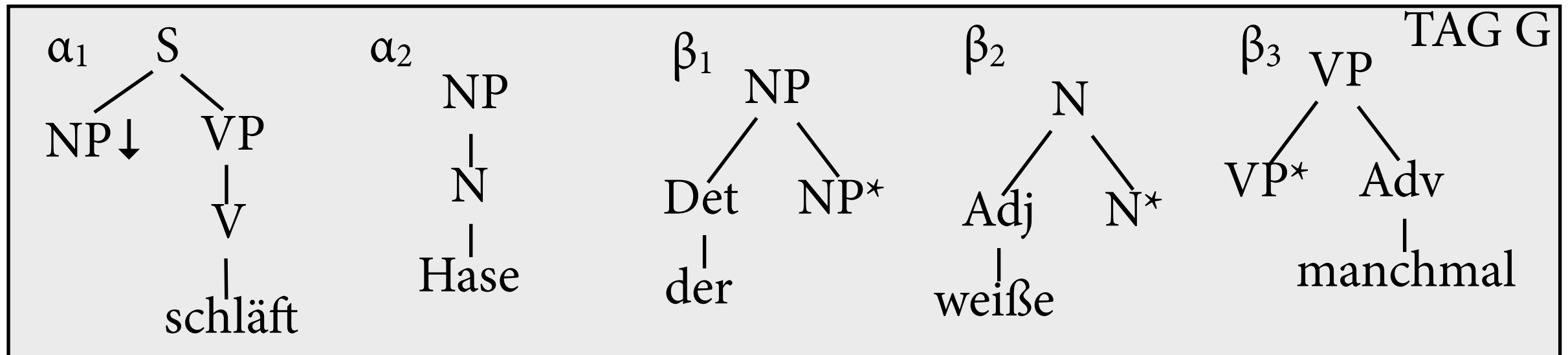
Vorlesung “Grammatikformalismen”  
Alexander Koller

25. April 2019

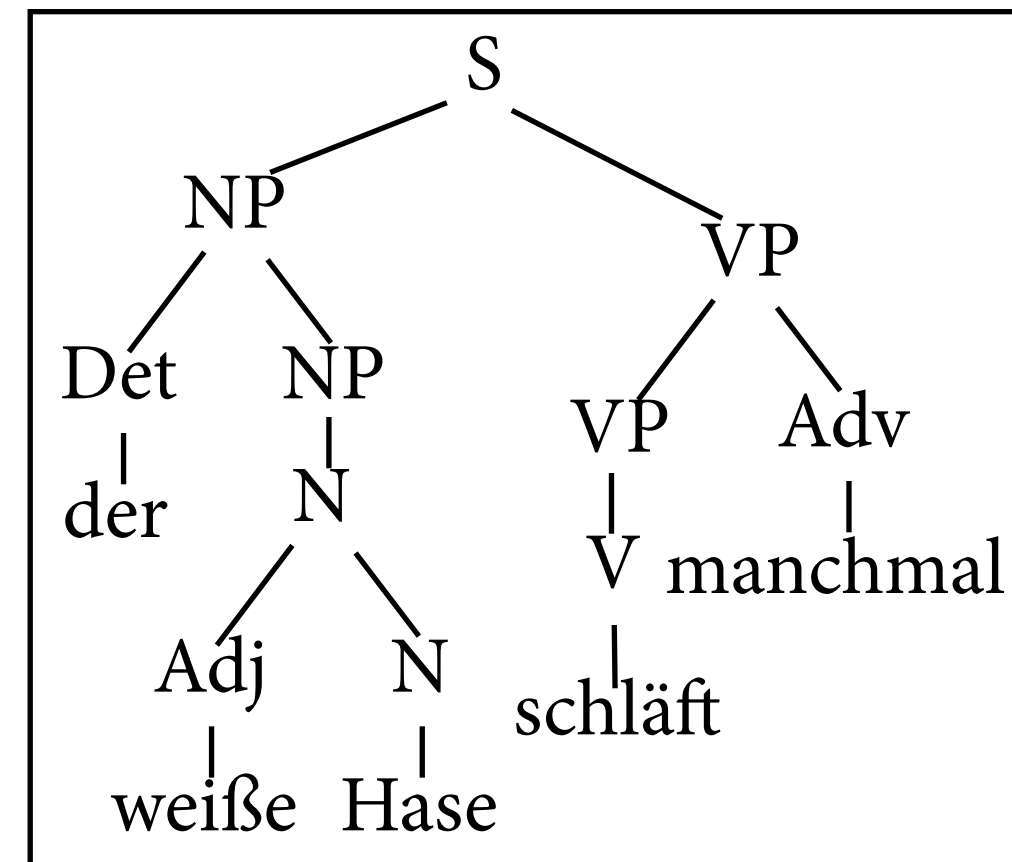
# Komplexität vs. Expressivität



# Ein Beispiel



Ableitungsbaum



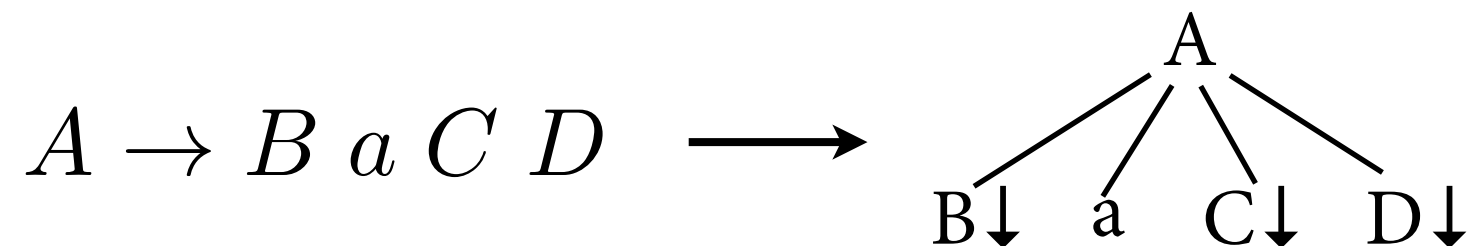
abgeleiteter Baum

# Übersicht

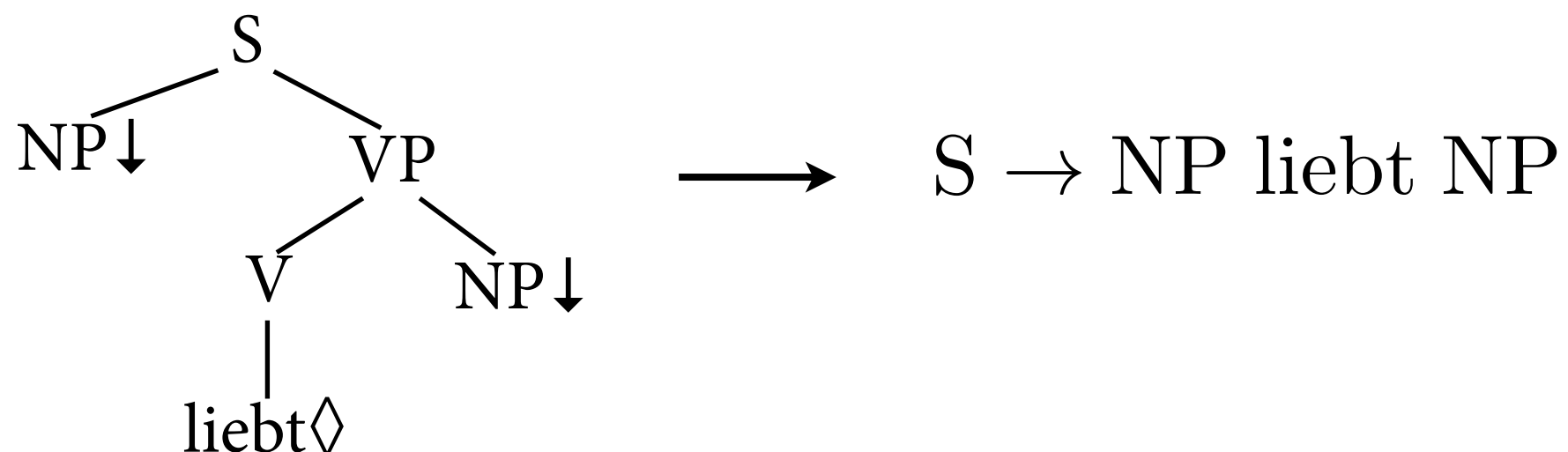
- Expressivität von TAG
- Parsing

# Expressivität von TAG

- Jede kfG kann man in stark äquivalente TSG übersetzen (nicht unbedingt lexikalisiert).



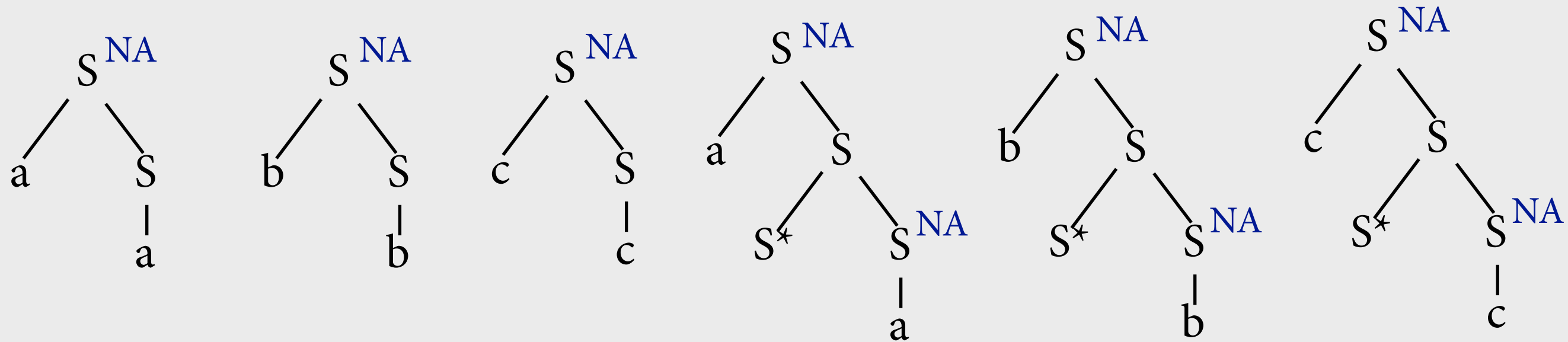
- Jede TSG kann man in schwach äquivalente kfG übersetzen.



- Das heißt: TSG ist schwach kontextfrei.

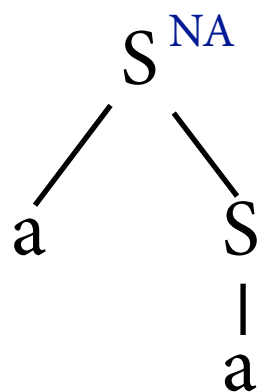
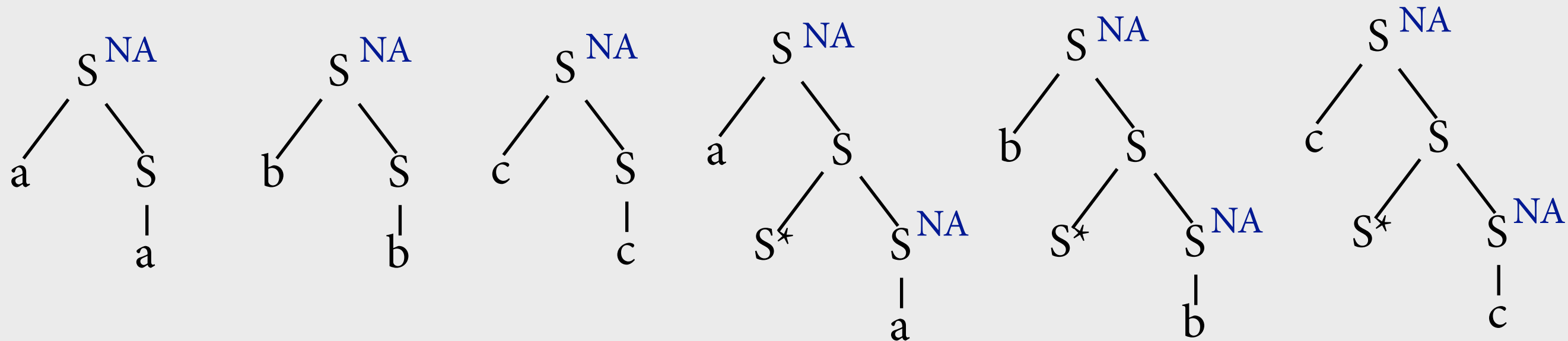
# TAG für die Copy-Sprache

$$\text{COPY} = \{ ww \mid w \in \Sigma^* \}$$



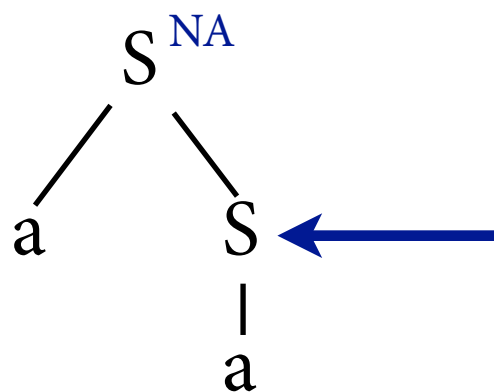
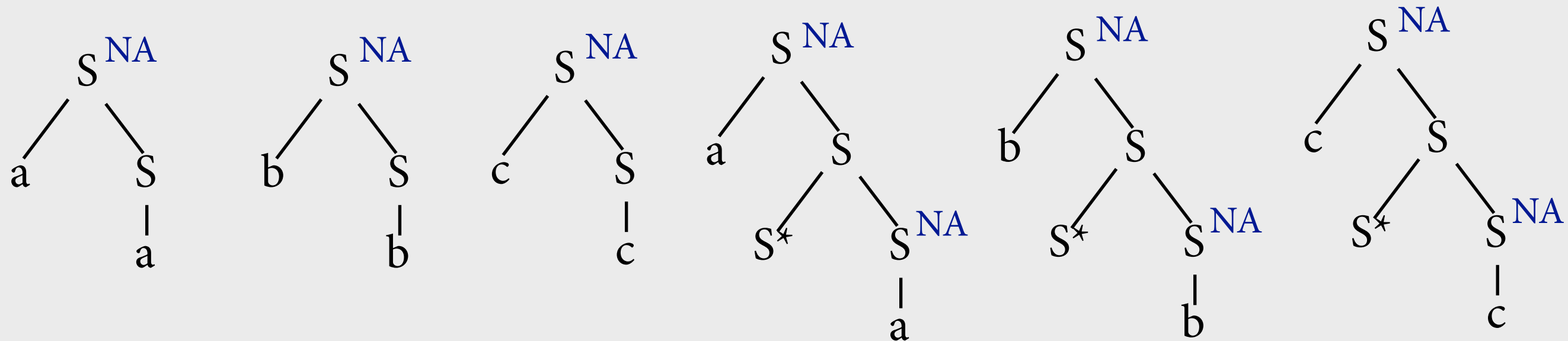
# TAG für die Copy-Sprache

$$\text{COPY} = \{ ww \mid w \in \Sigma^* \}$$



# TAG für die Copy-Sprache

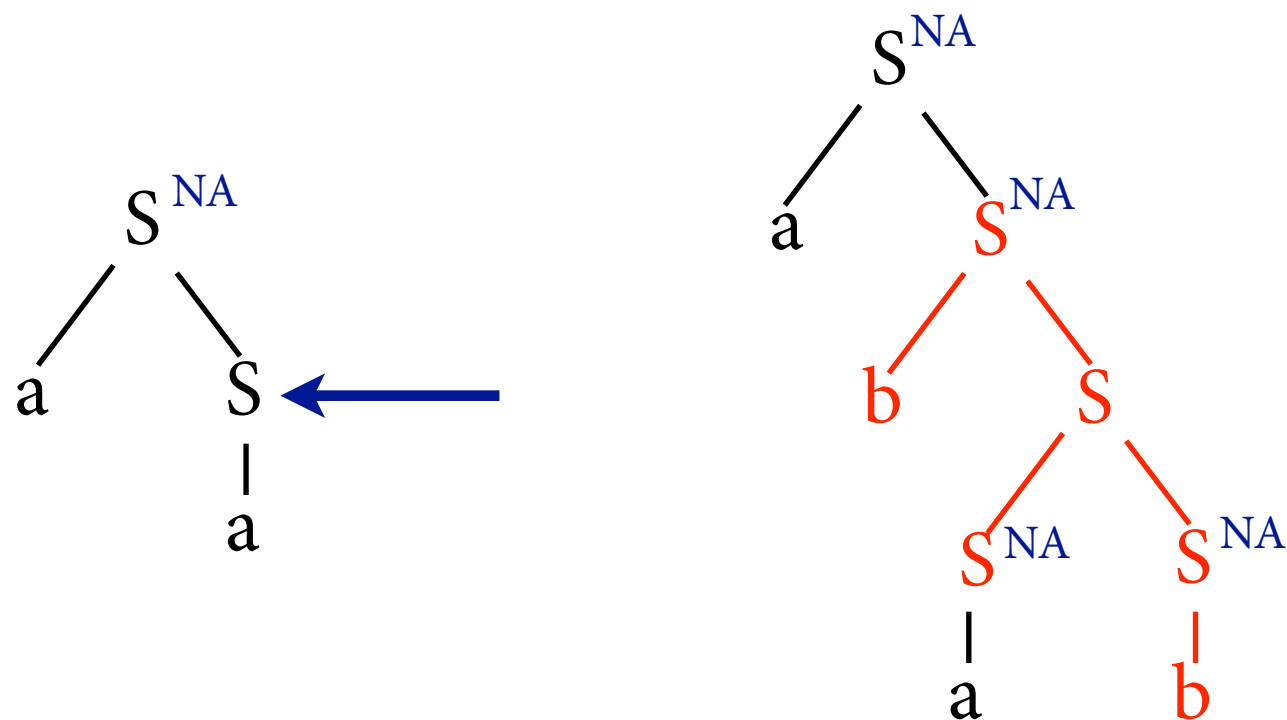
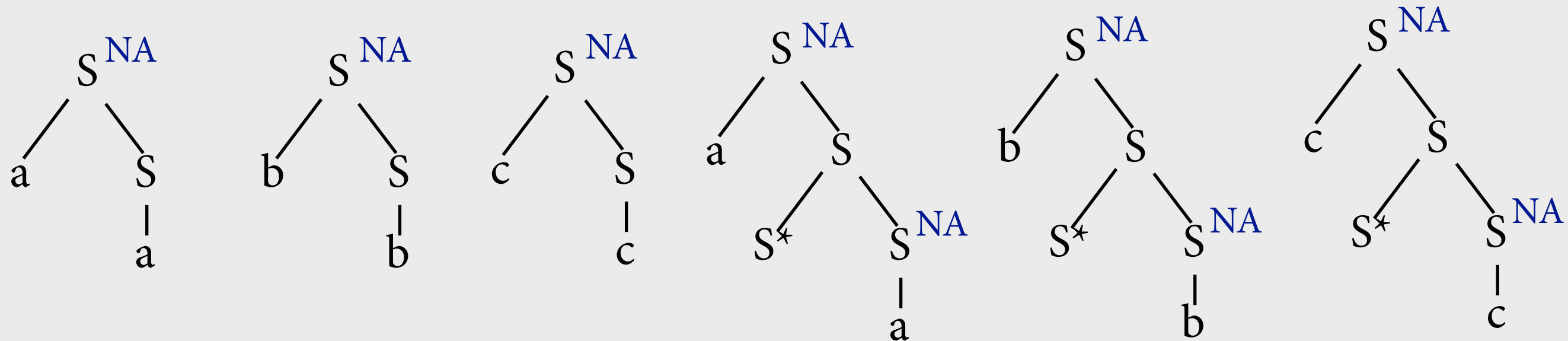
$$\text{COPY} = \{ ww \mid w \in \Sigma^* \}$$





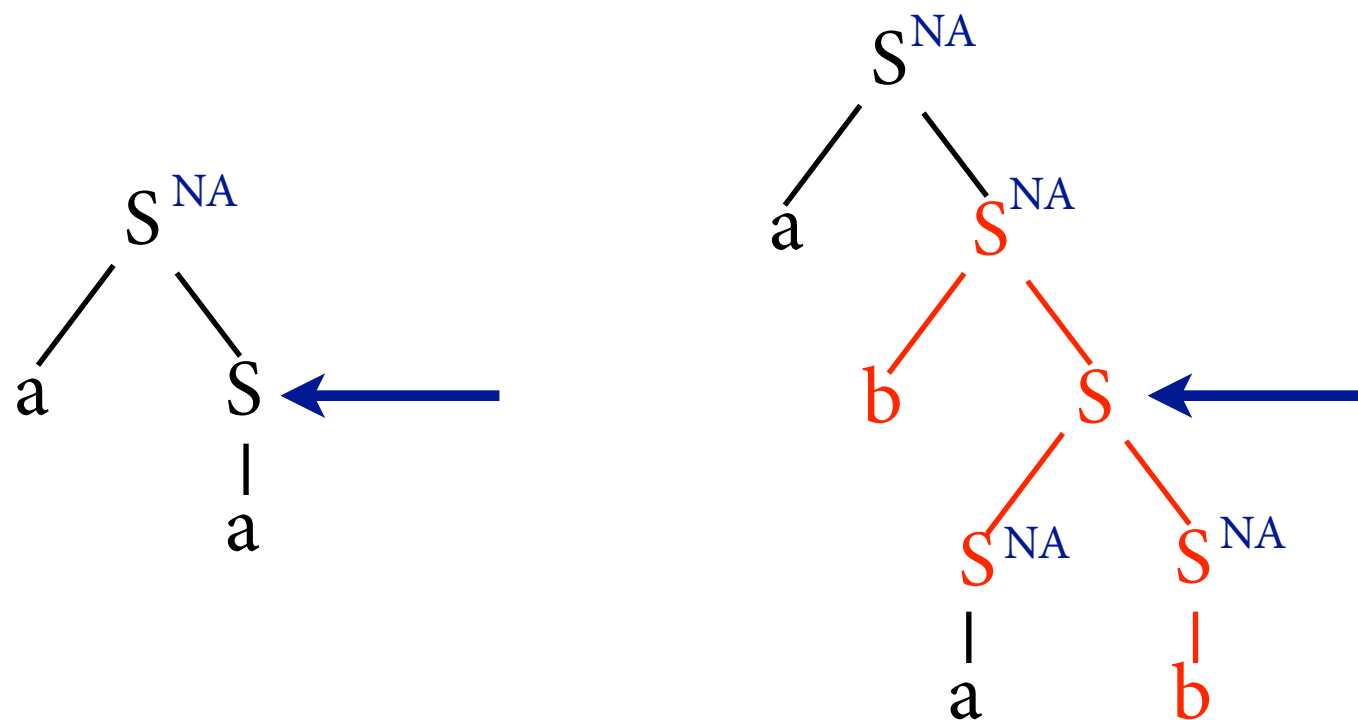
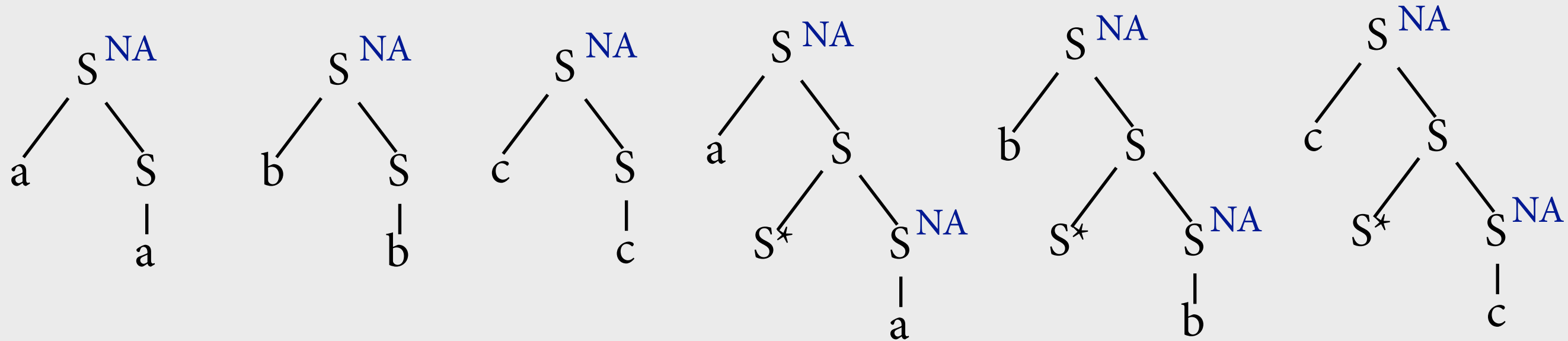
# TAG für die Copy-Sprache

$$\text{COPY} = \{ ww \mid w \in \Sigma^* \}$$



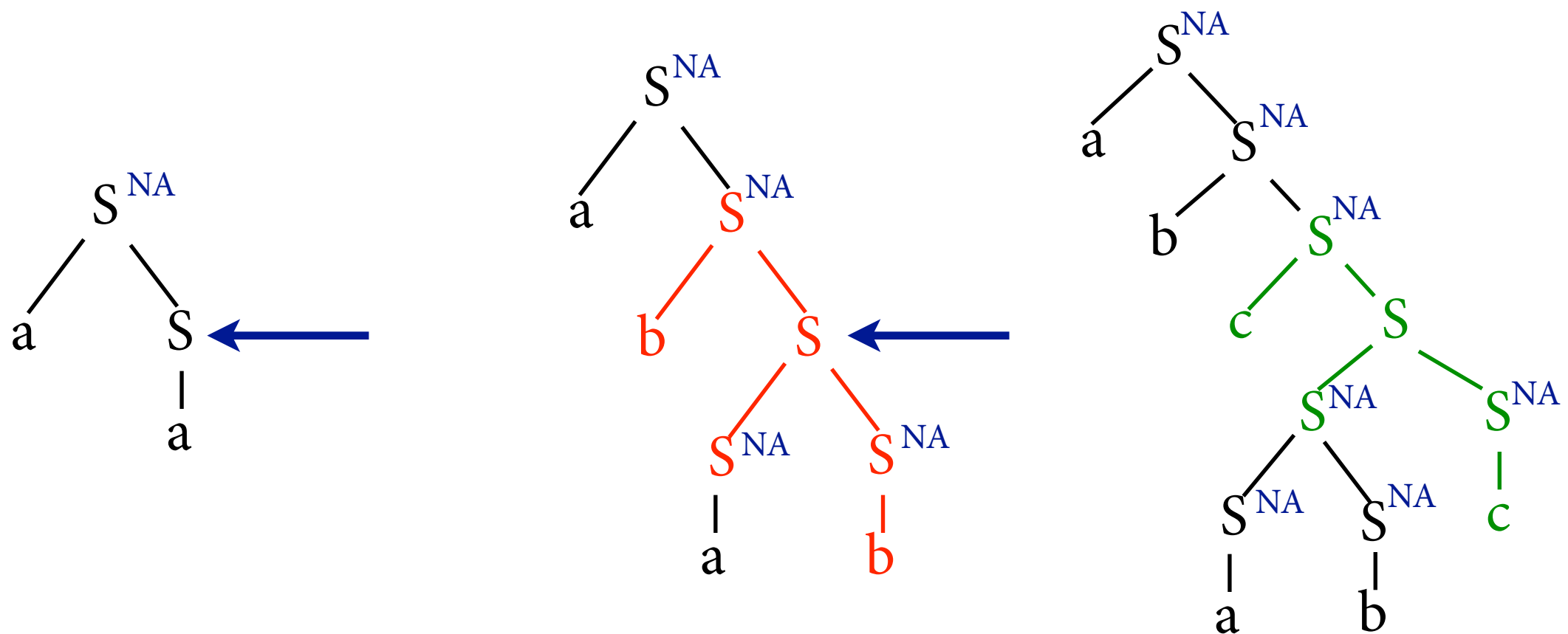
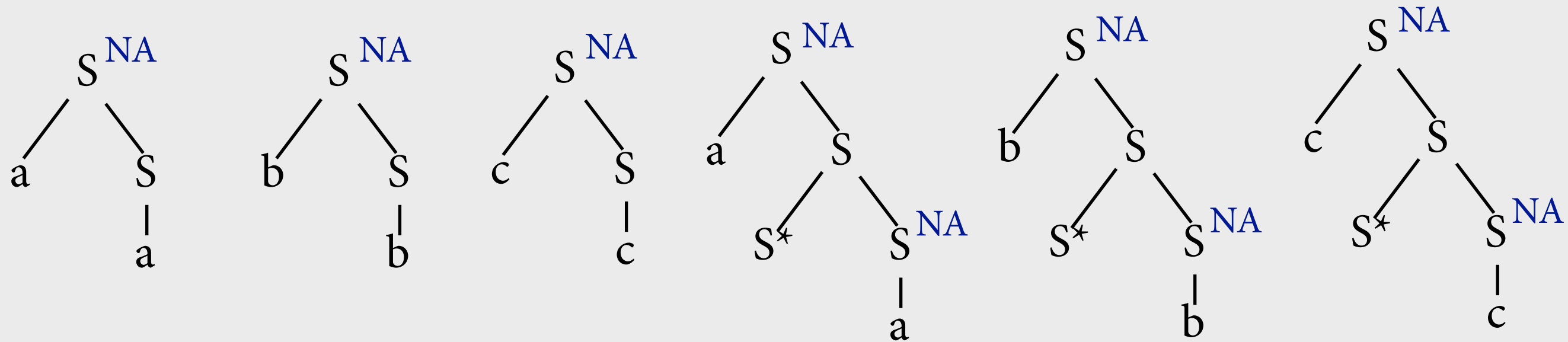
# TAG für die Copy-Sprache

$$\text{COPY} = \{ ww \mid w \in \Sigma^* \}$$



# TAG für die Copy-Sprache

$$\text{COPY} = \{ ww \mid w \in \Sigma^* \}$$



# KfG für COUNT(2)

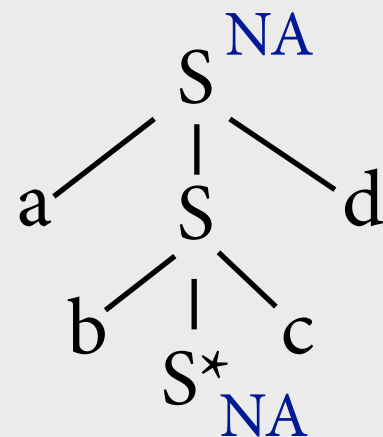
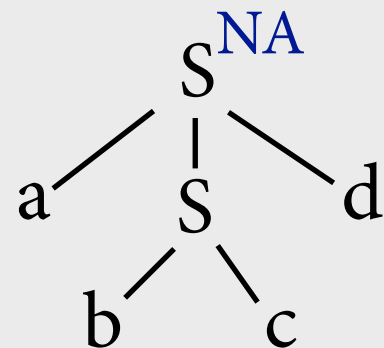
$$\text{COUNT}(2) = \{ a^n b^n \mid n \geq 1 \}$$

$$S \rightarrow a S b \mid a b$$

- KfGs können an einer einzigen Stelle den String “aufpumpen”. Die neuen Terminale müssen alle beieinander stehen.
- TAG: Mit Adjunktion den *Baum* aufpumpen. Neue Blätter müssen im String nicht nahe beieinander stehen.

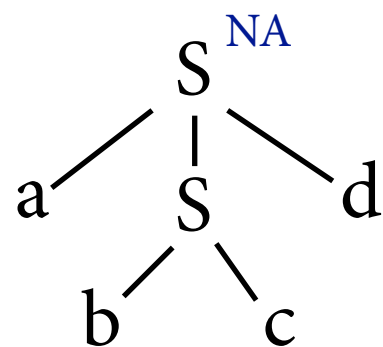
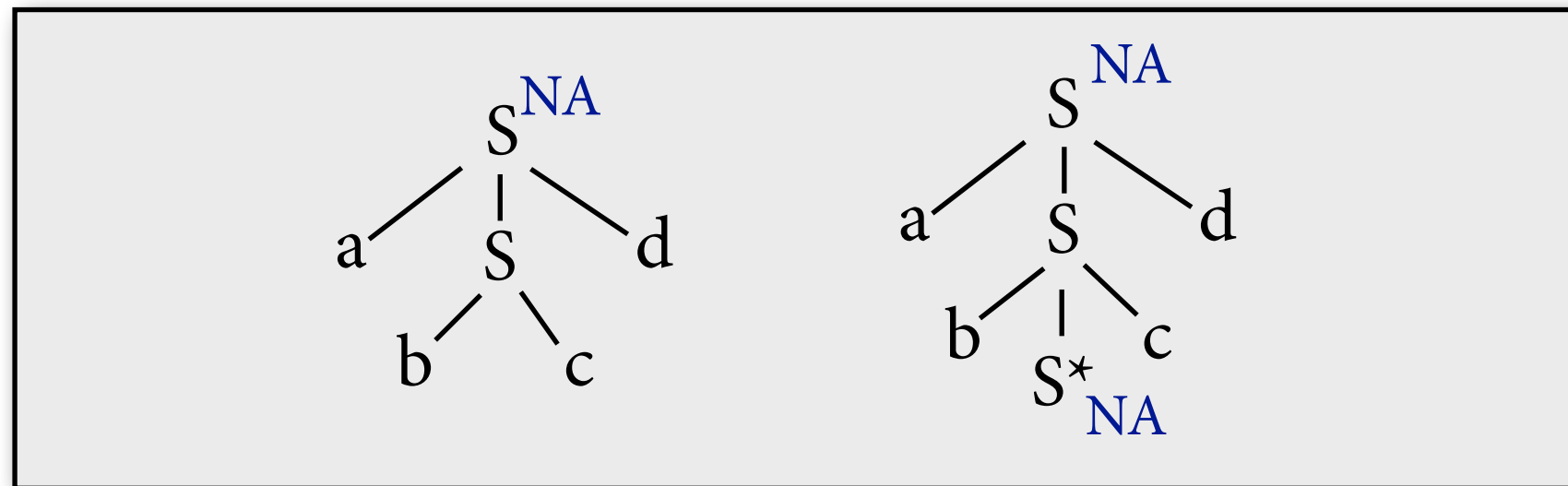
# TAG für COUNT(4)

$$\text{COUNT}(4) = \{ a^n b^n c^n d^n \mid n \geq 1 \}$$



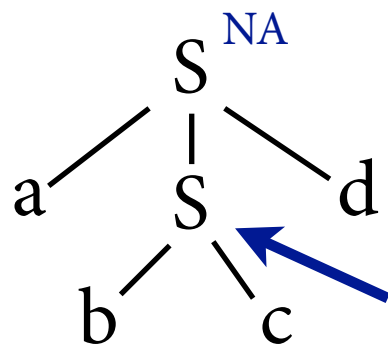
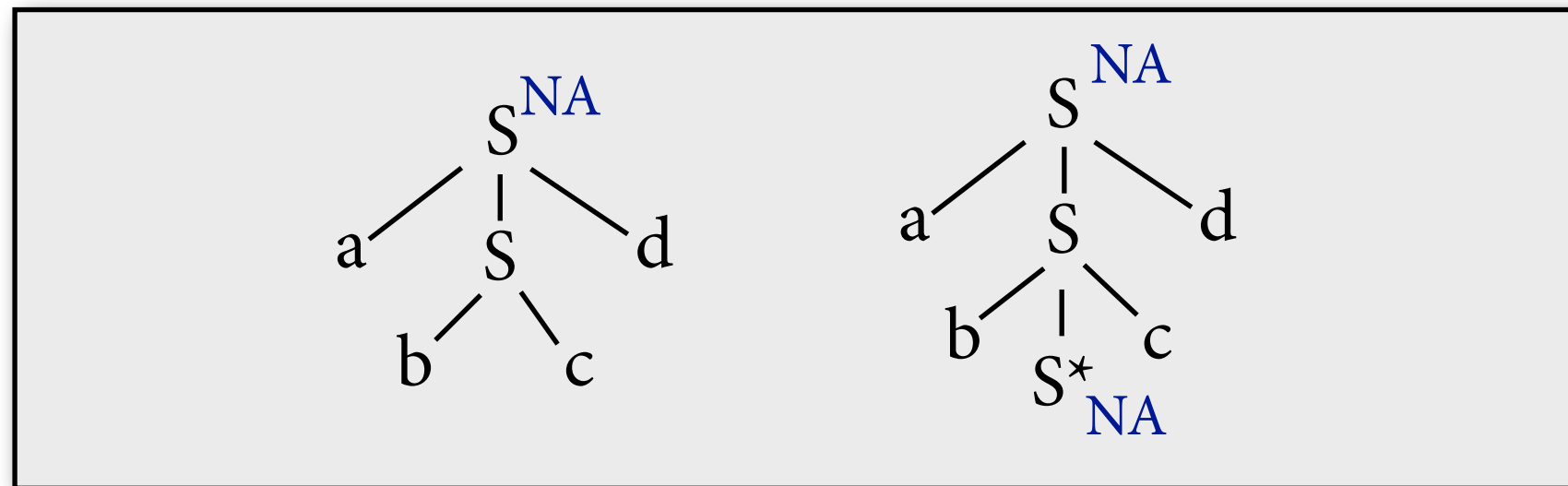
# TAG für COUNT(4)

$$\text{COUNT}(4) = \{ a^n b^n c^n d^n \mid n \geq 1 \}$$



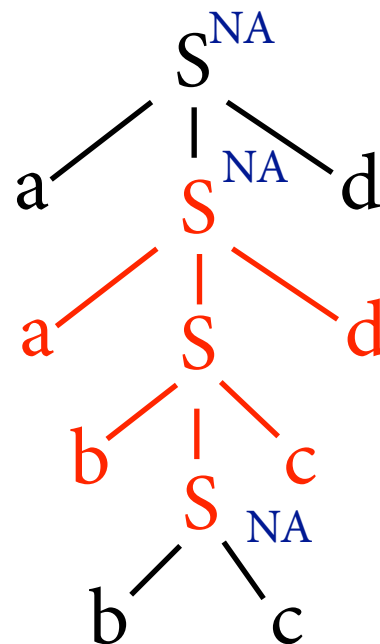
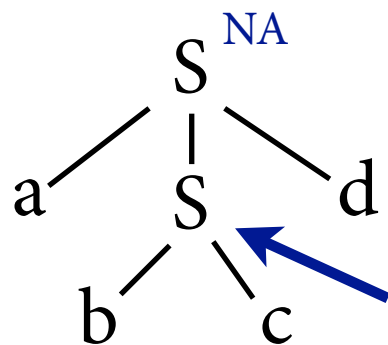
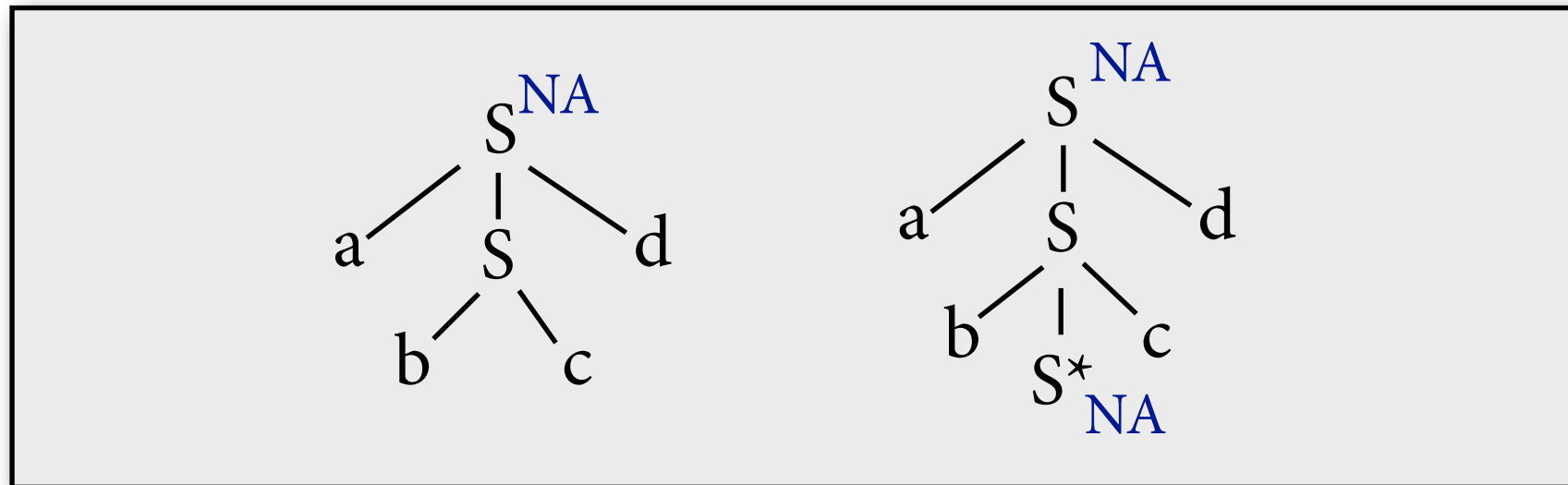
# TAG für COUNT(4)

$$\text{COUNT}(4) = \{ a^n b^n c^n d^n \mid n \geq 1 \}$$



# TAG für COUNT(4)

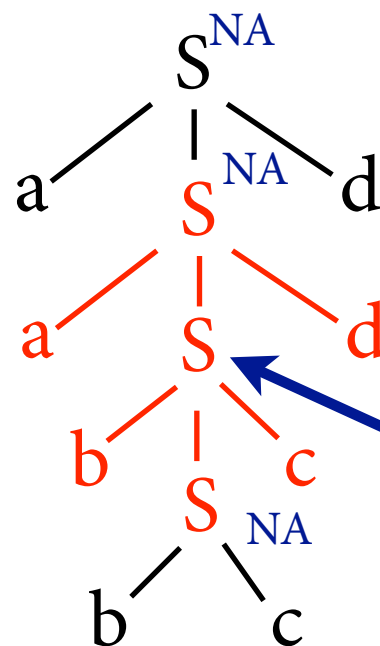
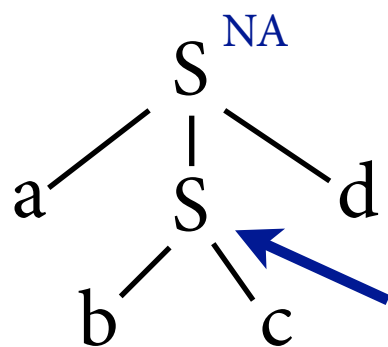
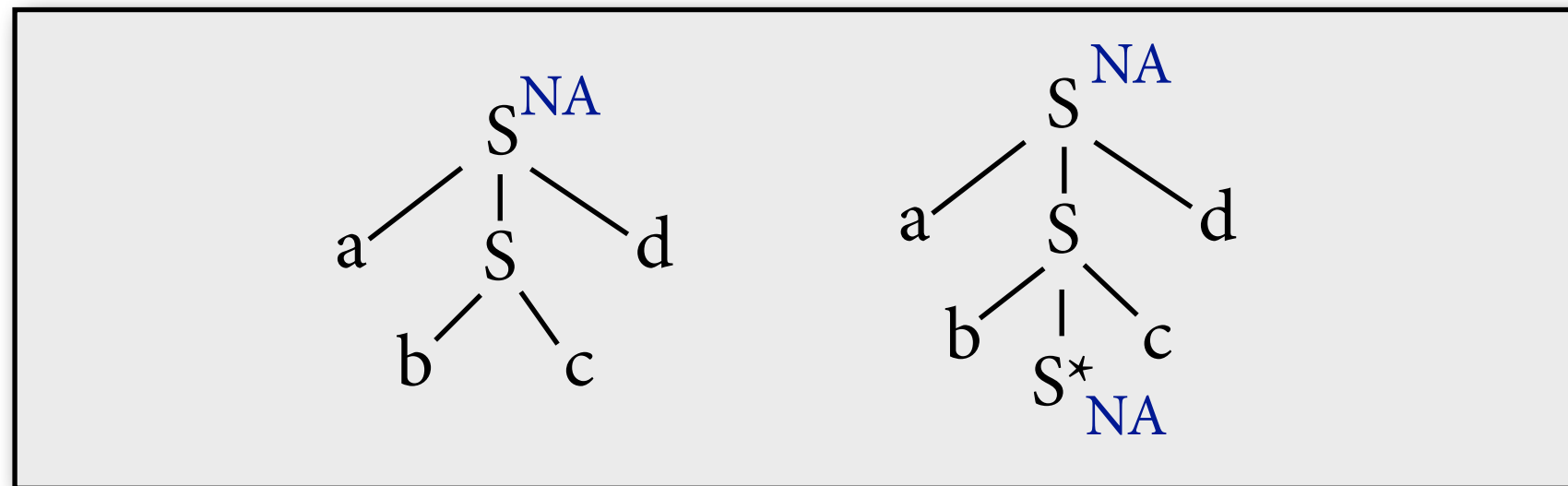
$$\text{COUNT}(4) = \{ a^n b^n c^n d^n \mid n \geq 1 \}$$





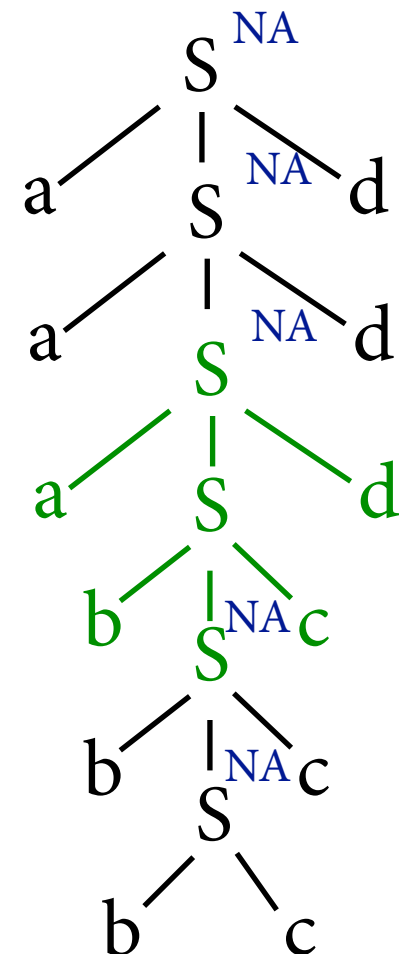
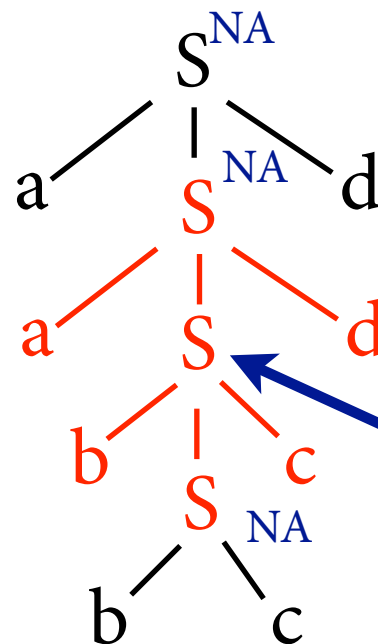
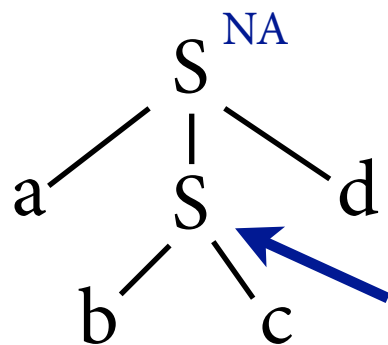
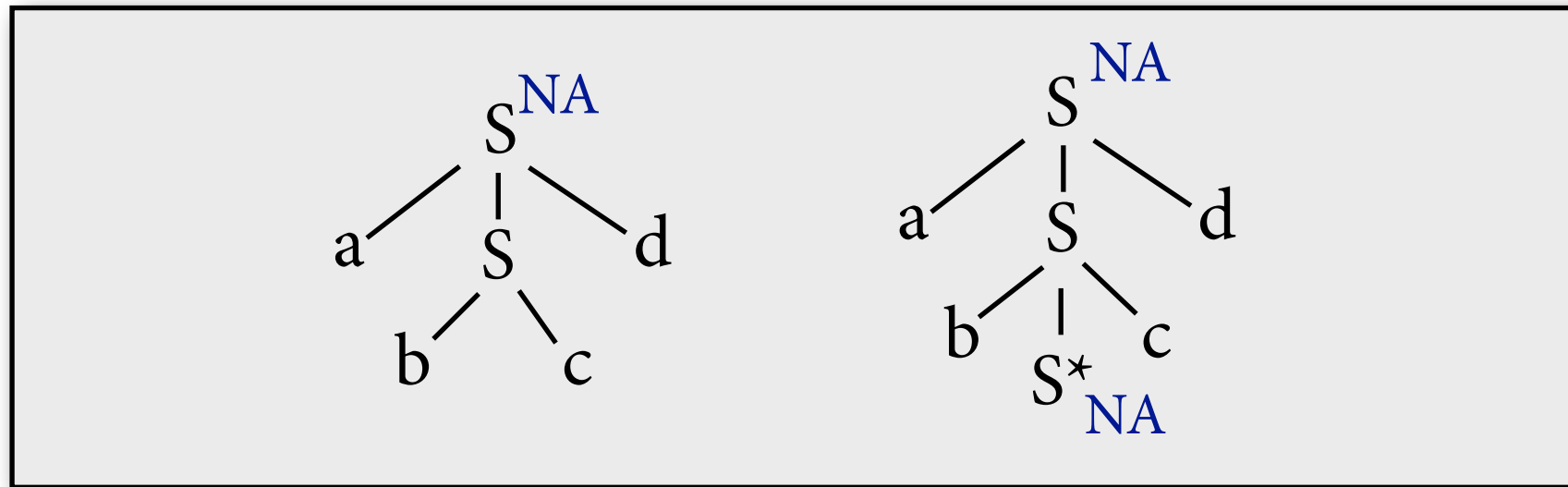
# TAG für COUNT(4)

$$\text{COUNT}(4) = \{ a^n b^n c^n d^n \mid n \geq 1 \}$$



# TAG für COUNT(4)

$$\text{COUNT}(4) = \{ a^n b^n c^n d^n \mid n \geq 1 \}$$



# Expressivität von TAG

- TAG kann mehr Sprachen beschreiben als kfG, insbesondere:
  - ▶ Fernabhängigkeiten ( $\approx$  COUNT(4))
  - ▶ Cross-serial dependencies ( $\approx$  COPY)
- TAG kann aber nicht alle kontextsensitiven Sprachen beschreiben.
  - ▶ Z.B. nicht COUNT(5); kann man mit TAG-Variante des Pumping-Lemmas zeigen.
  - ▶ Idee: TAG kann *zwei* Stellen im String gleichzeitig aufpumpen, aber nicht drei.

# Parsing von TAG

- Wenn TAG expressiver ist als kfG, kann man dann noch effizient parsen?
- Antwort: es geht polynomiell. Hier: CKY-Parser.
- CKY-Parser verlangt *binäre* TAG-Grammatiken: Jeder Knoten in jedem Elementarbaum hat höchstens zwei Kinder.

# CKY-Parser für kfG

- Parsing für kfGs in Chomsky-Normalform:  
CKY-Algorithmus (Cocke-Younger-Kasami).
- Grundidee: Leite *Items* der Form  $[A, i, k]$  ab.  
Bedeutet: Kann Substring  $w_i \dots w_{k-1}$  aus  $A$  ableiten.
- String in Sprache = CKY leitet  $[S, 1, n+1]$  ab.

$$\frac{A \rightarrow w_i \text{ in Grammatik}}{[A, i, i+1]}$$

$$\frac{[B, i, j] \quad [C, j, k] \quad A \rightarrow B C \text{ in Grammatik}}{[A, i, k]}$$

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V \ NP$

Hans      isst      ein      Käsebrod

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V \ NP$

Hans isst ein Käsebrot.

Hans	isst	ein	Käsebrot

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

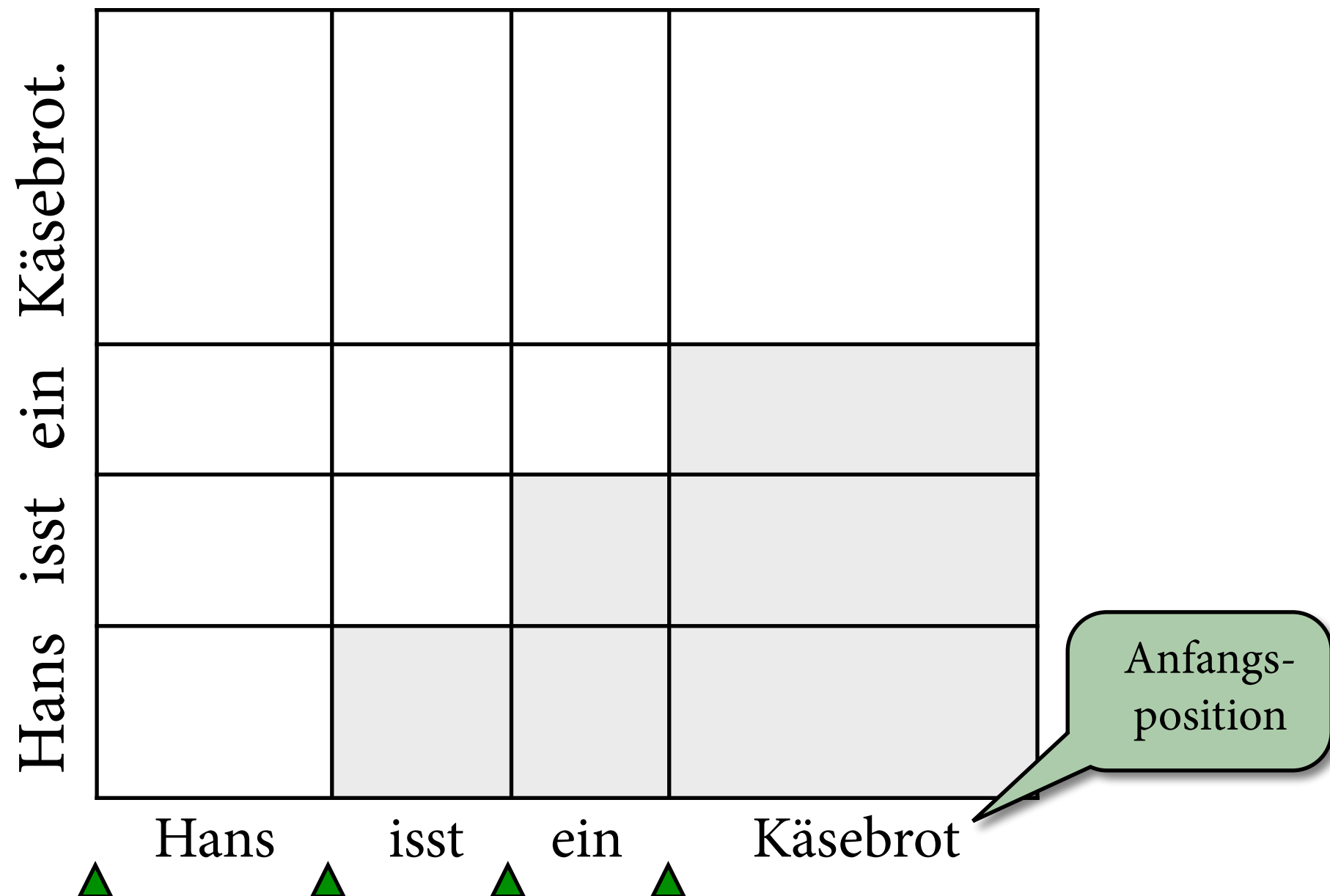
$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V \ NP$





# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

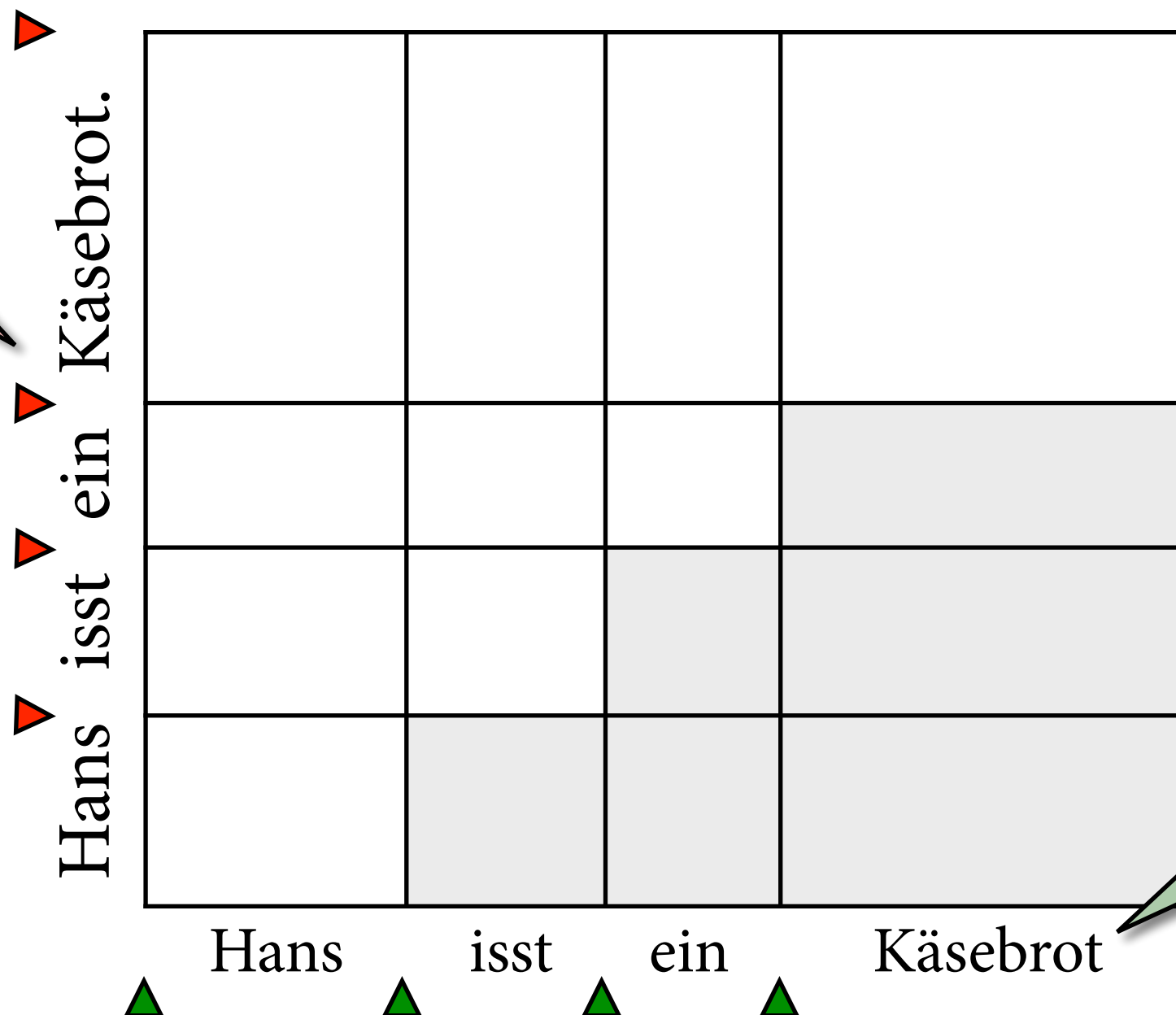
$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V \ NP$

Endposition



Anfangs-  
position

# Der CKY-Parser

$$S \rightarrow NP \quad VP$$

$V \rightarrow$  isst

Det  $\rightarrow$  ein

NP  $\rightarrow$  Det N

NP → Hans

# N $\rightarrow$ Käsebrot

$$VP \rightarrow V \ NP$$

# Chart

## Endposition

Hans isst ein Käsebröt.

# Anfangs- position

Hans

isst

ein

# Käsebrot

# Der CKY-Parser

$$S \rightarrow NP \quad VP$$

$V \rightarrow$  isst

Det  $\rightarrow$  ein

NP  $\rightarrow$  Det N

NP → Hans

# N $\rightarrow$ Käsebrot

$$VP \rightarrow V \ NP$$

# Chart

## Endposition

Hans isst ein Käsebröt.

# Anfangs- position

# Hans

isst

ein

# Käsebrot

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V \ NP$

Chart

Endposition

Hans isst ein Käsebrot.				
		V		
	NP			
	Hans	isst	ein	Käsebrot

Anfangs-  
position

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V \ NP$

Chart

Endposition

Hans isst ein Käsebrot.				
			Det	
		V		
	NP			
	Hans	isst	ein	Käsebrot

Anfangs-  
position

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V \ NP$

Chart

Endposition

Hans isst ein Käsebrot.				N
			Det	
		V		
	NP			
	Hans	isst	ein	Käsebrot

Anfangs-  
position

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V \ NP$

Chart

Endposition

Hans isst ein Käsebrot.			NP	N
			Det	
		V		
	NP			
	Hans	isst	ein	Käsebrot

Anfangs-  
position

# Der CKY-Parser

$S \rightarrow NP \ VP$        $V \rightarrow \text{isst}$        $\text{Det} \rightarrow \text{ein}$   
 $NP \rightarrow \text{Det} \ N$        $NP \rightarrow \text{Hans}$        $N \rightarrow \text{Käse}$   
 $VP \rightarrow V \ NP$        $N \rightarrow \text{brot}$

Chart

Endposition

Hans isst ein Käse <b>rot</b> .			NP	N
			Det	
		V		
	NP			
	Hans	isst	ein	Käse <b>rot</b>

Anfangs-  
position



# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

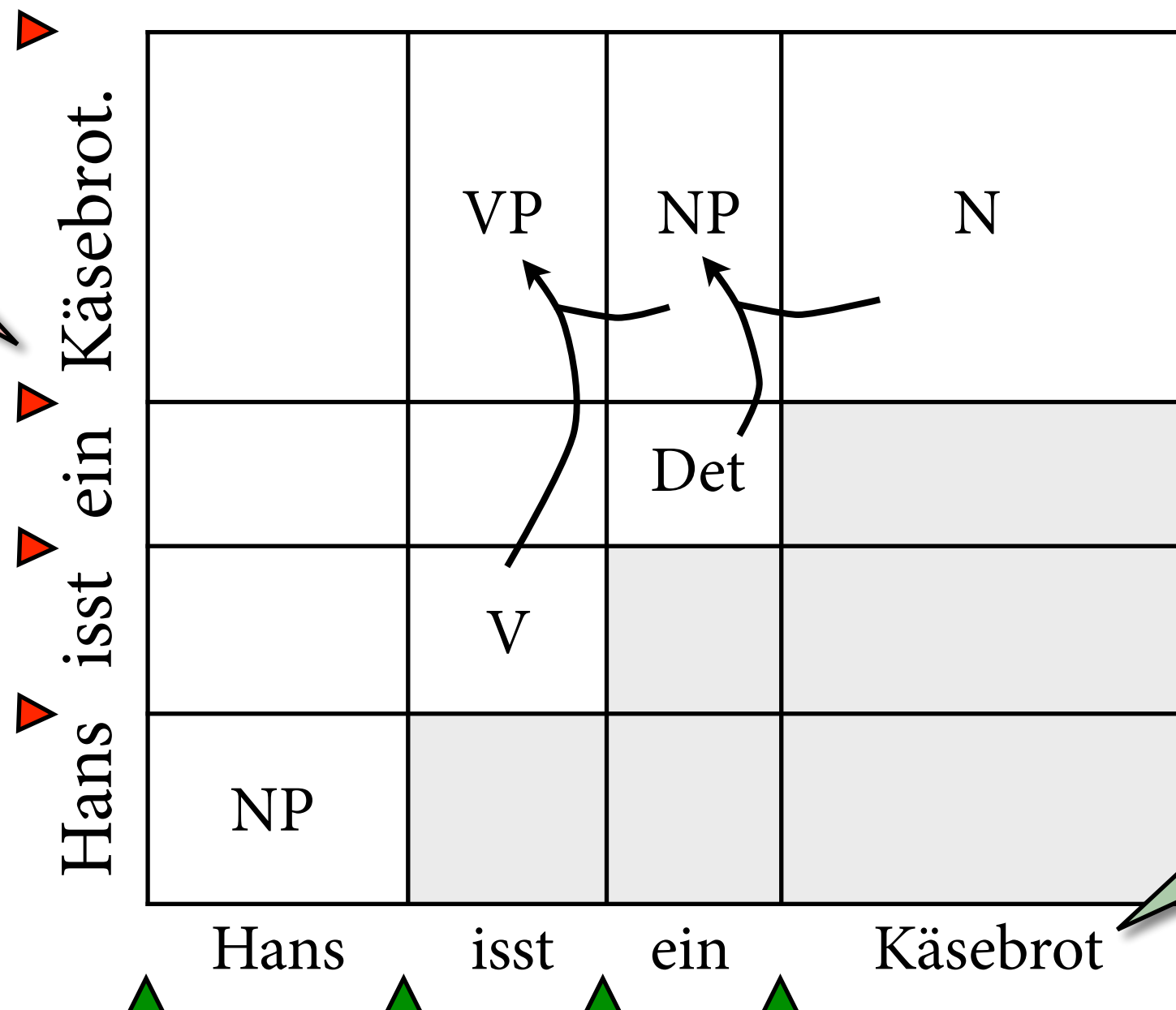
$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V \ NP$

Chart

Endposition



Anfangs-  
position

# Der CKY-Parser

$S \rightarrow NP \ VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det} \ N$

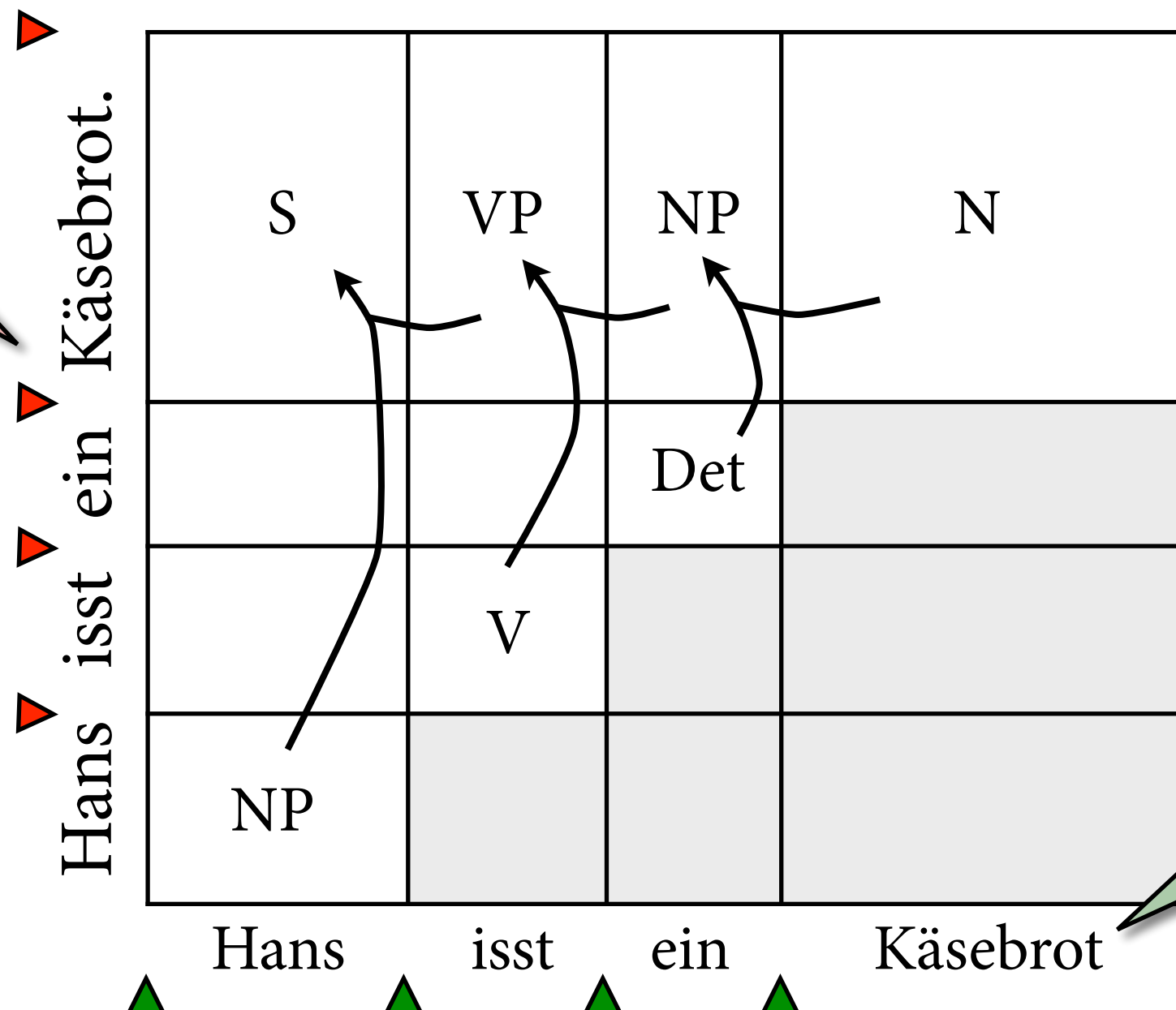
$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V \ NP$

Chart

Endposition



Anfangs-  
position

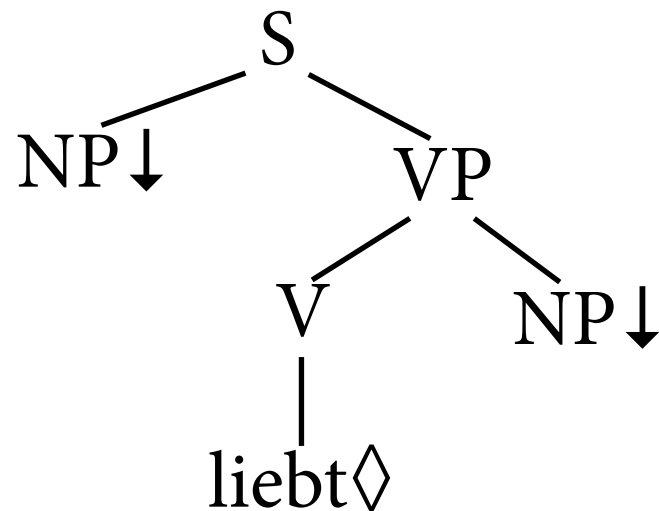
# Laufzeit von CKY

- Faustregel für Laufzeit von Parsing-Schema:  
wenn Regeln max.  $k$  unabhängige Variablen für  
Stringpositionen haben, ist Laufzeit  $O(n^k)$ .
- Laufzeit von CKY ist also  $O(n^3)$ .

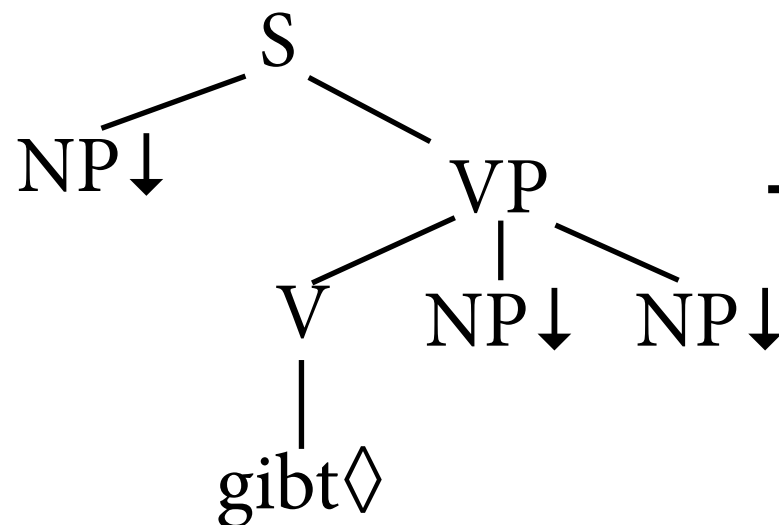
$$\frac{A \rightarrow w_i \text{ in Grammatik}}{[A, i, i+1]} \quad \frac{[B, i, j] \quad [C, j, k] \quad A \rightarrow B C \text{ in Grammatik}}{[A, i, k]}$$

# Parsing von TSG: 1. Ansatz

- Konvertiere jeden Elementarbaum als Ganzes in schwach äquivalente kfG-Regel:



$S \rightarrow \text{NP liebt NP}$



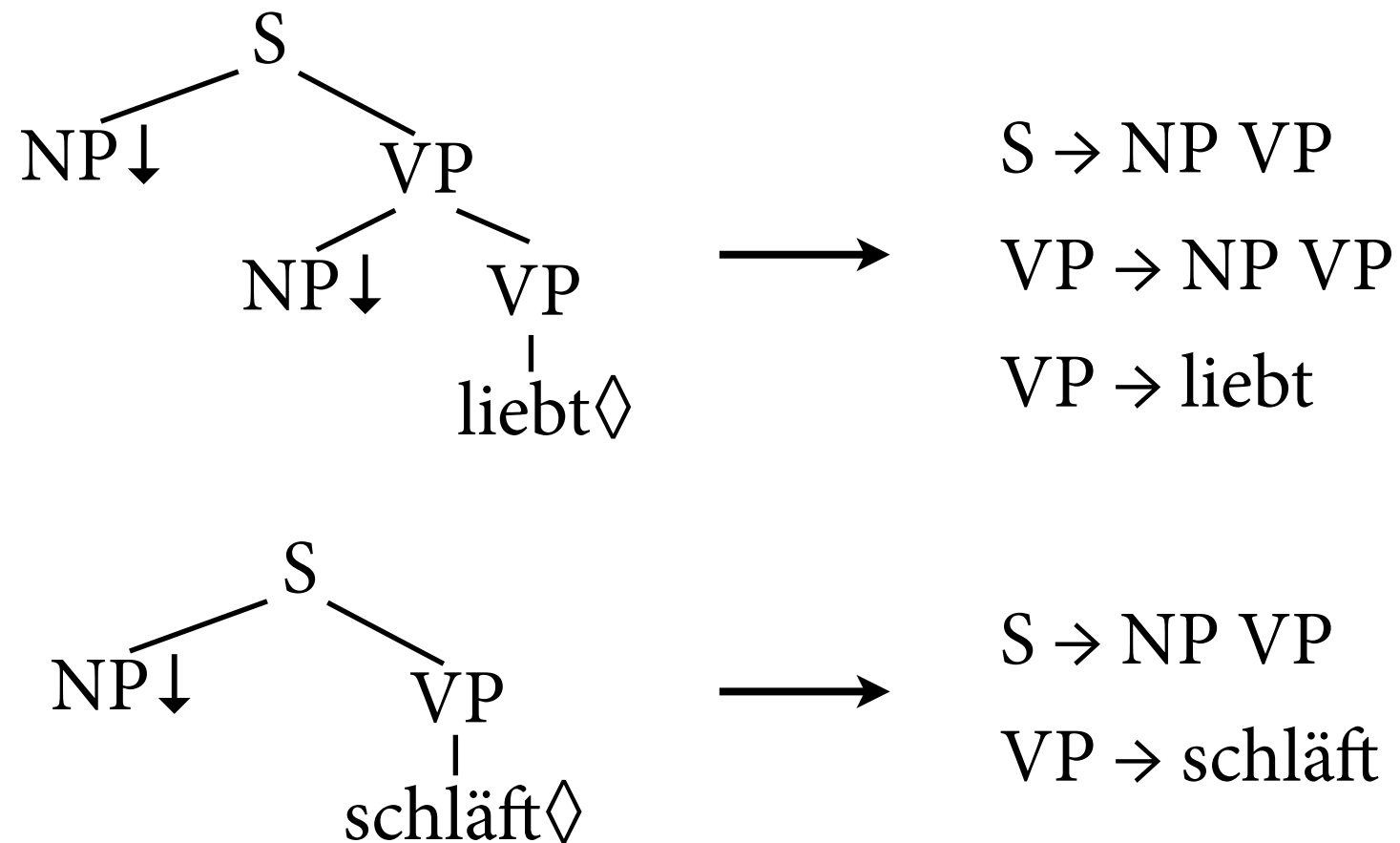
$S \rightarrow \text{NP gibt NP NP}$

# TSG-Parsing, 2. Ansatz

- Problem: kfG ist nicht in Chomsky-Normalform, insbesondere: mehr als zwei Nichtterminale auf rechter Regelseite → CKY nicht direkt anwendbar.
- Grundidee aus CNF-Transformation für kfGen:  
*Binarisierung*
  - ▶ r-stellige Regel in r-1 zweistellige Regeln aufbrechen
  - ▶ damit sinkt Rang r auf 2  $\Rightarrow$  Laufzeit  $O(n^{r+1}) = O(n^3)$
- Für TSG mit binären Elementarbäumen bietet es sich an, entlang der Baumstruktur aufzubrechen.

# Gut aufpassen

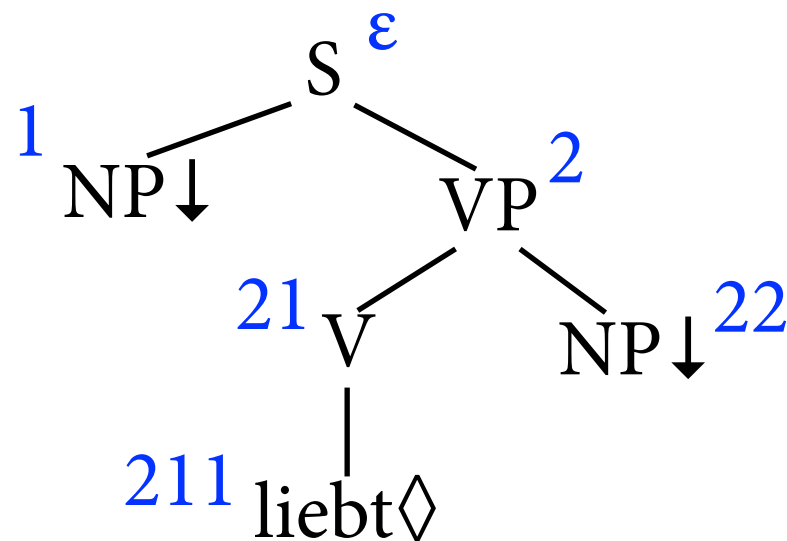
- Die naive Lösung klappt nicht:



- Man könnte dann Teile verschiedener Elementarbäume miteinander kombinieren.

# Pfade in Bäumen

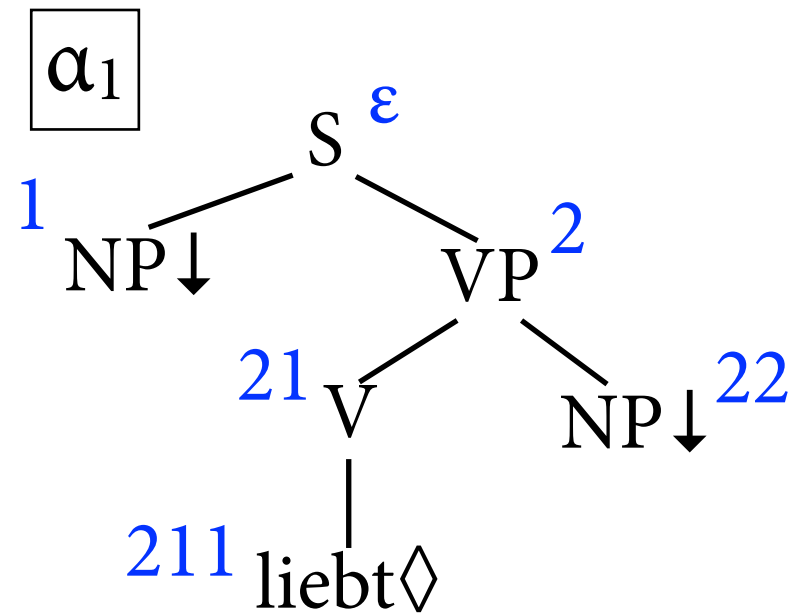
- Man kann jeden Knoten in einem Baum mit einem Wort aus  $N^*$  beschreiben, das sagt, wie man von der Wurzel aus hinkommt.



- Diese Wörter heißen manchmal auch *Gorn-Adressen*.

# Buchführen über Bäume

Lösung: eigene NT-Symbole für jeden Elementarbaum.



$$S \rightarrow NP_1^{\alpha_1} VP_2^{\alpha_1}$$

$$VP_2^{\alpha_1} \rightarrow V_{21}^{\alpha_1} NP_{22}^{\alpha_1}$$

$$V_{21}^{\alpha_1} \rightarrow \text{liebt}$$

$$NP_1^{\alpha_1} \rightarrow NP \quad NP_{22}^{\alpha_1} \rightarrow NP$$

Bsp. CKY-Regelinstanz:

$$\frac{[V_{21}^{\alpha_1}, i, j] \quad [NP_{22}^{\alpha_1}, j, k]}{[VP_2^{\alpha_1}, i, k]}$$

Allgemein:

$$\frac{[B_{\pi 1}^{\alpha}, i, j] \quad [C_{\pi 2}^{\alpha}, j, k] \quad \text{label}(\alpha, \pi) = A}{[A_{\pi}^{\alpha}, i, k]}$$



# CKY-Parser für TSG

- Statt in Nichtterminalen kann man sich die Knoten auch direkt in den Parse-Items merken.

$$\frac{[B, i, j, \alpha, \pi 1] \quad [C, j, k, \alpha, \pi 2] \quad \text{label}(\alpha, \pi) = A}{[A, i, k, \alpha, \pi]}$$

$$\frac{[A, i, k, \alpha', \epsilon] \quad \text{label}(\alpha, \pi) = A \downarrow}{[A, i, k, \alpha, \pi]} \quad \frac{w_i = a \quad \text{label}(\alpha, \pi 1) = a \diamond}{[A, i, i + 1, \alpha, \pi]}$$

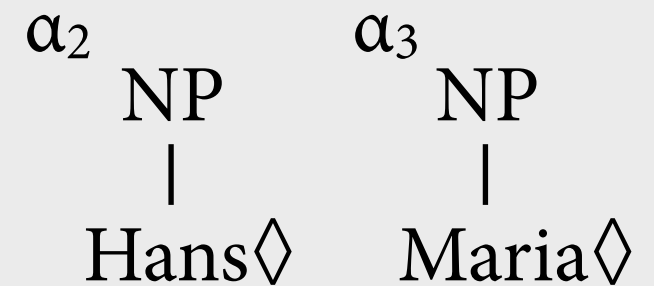
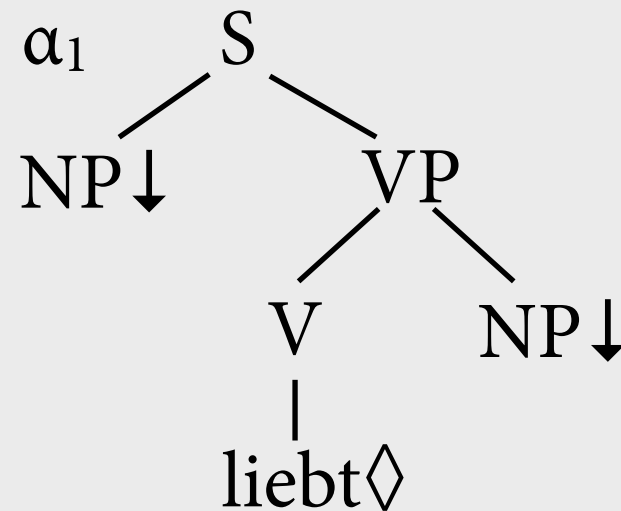
- Laufzeit:  $O(n^3)$ .

# Beispiel

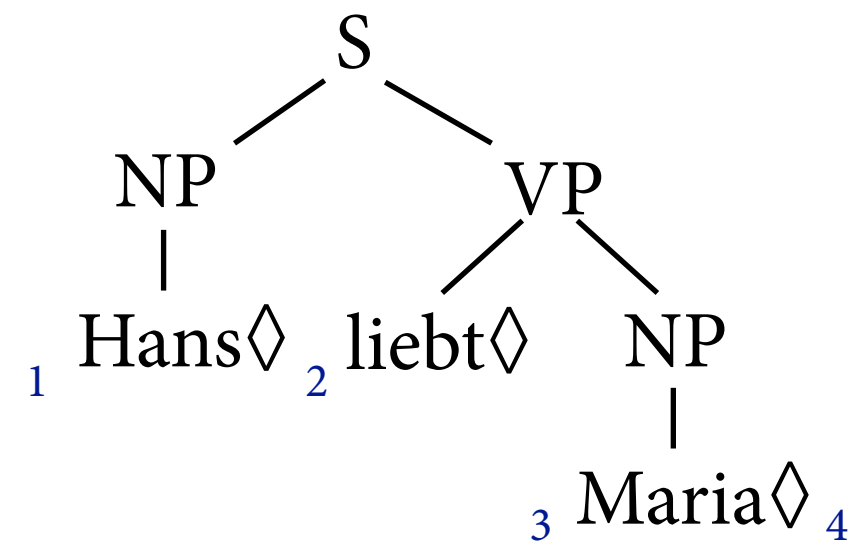
$$\frac{[B, i, j, \alpha, \pi 1] \quad [C, j, k, \alpha, \pi 2] \quad \text{label}(\alpha, \pi) = A}{[A, i, k, \alpha, \pi]}$$

$$\frac{[A, i, k, \alpha', \epsilon] \quad \text{label}(\alpha, \pi) = A \downarrow}{[A, i, k, \alpha, \pi]}$$

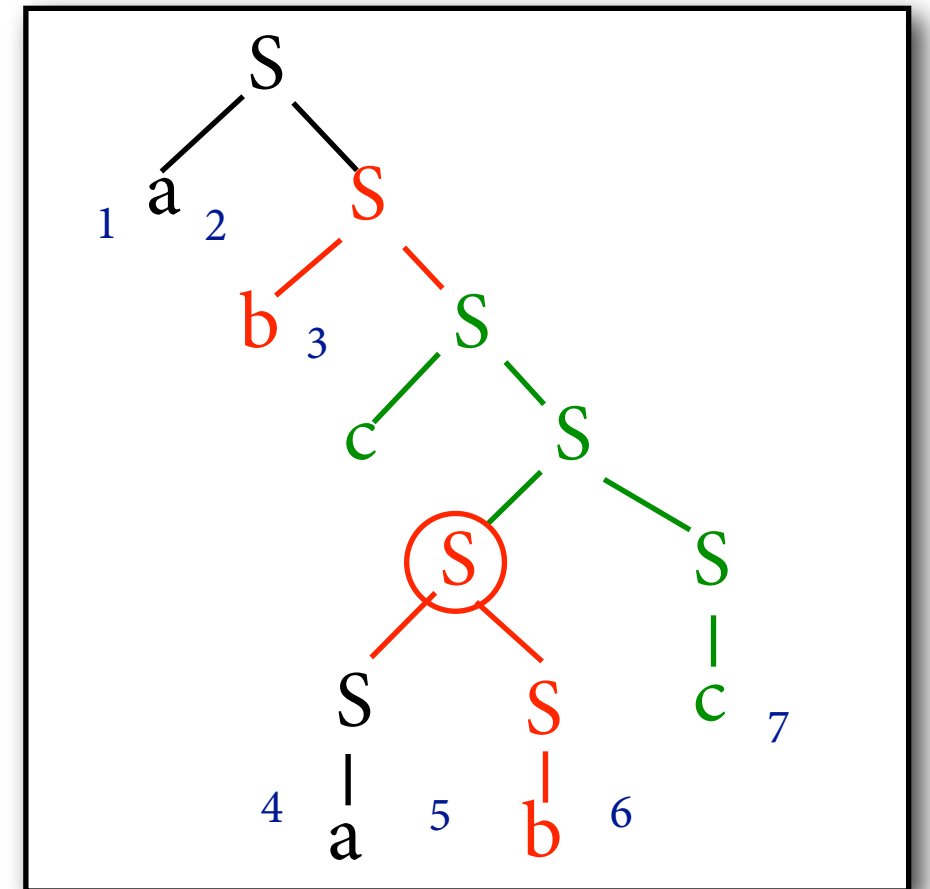
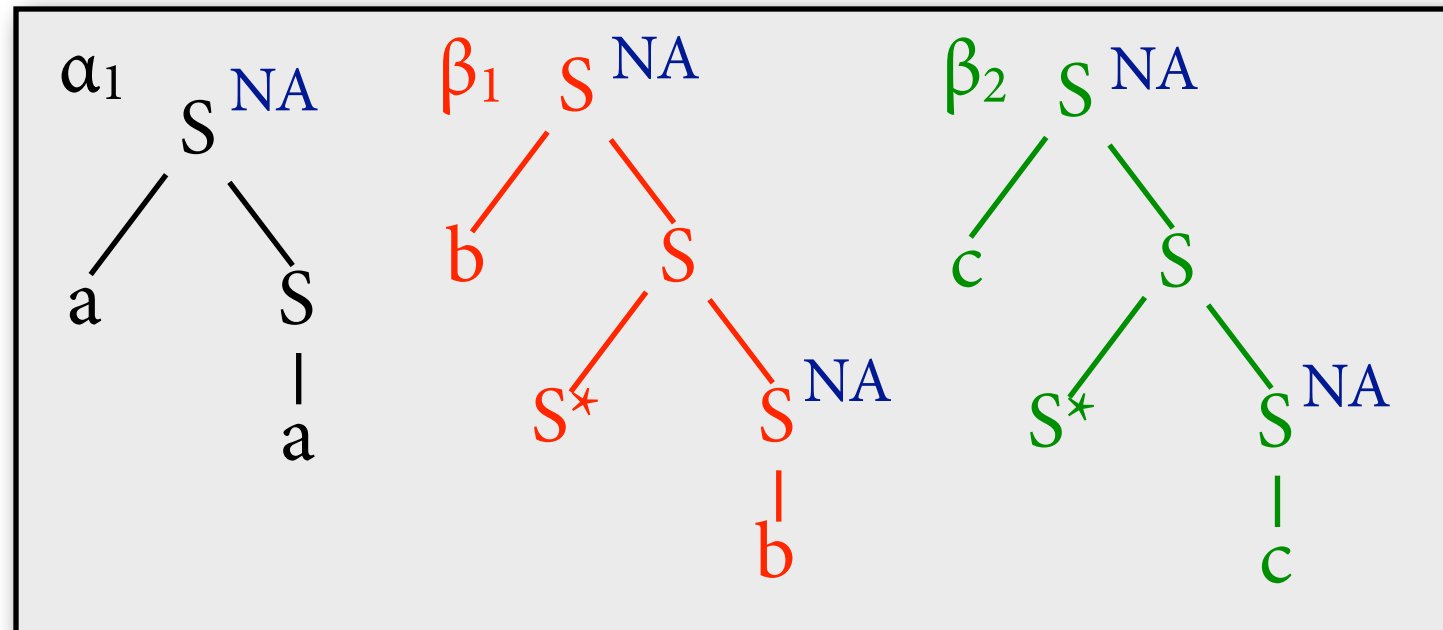
$$\frac{w_i = a \quad \text{label}(\alpha, \pi 1) = a \diamond}{[A, i, i + 1, \alpha, \pi]}$$



$w_1 = \text{Hans}$ $[\text{NP}, 1, 2, \alpha_2, \epsilon]$	$w_2 = \text{liebt}$ $[\text{NP}, 2, 3, \alpha_1, 21]$	$w_3 = \text{Maria}$ $[\text{NP}, 3, 4, \alpha_3, \epsilon]$
$[\text{NP}, 1, 2, \alpha_1, 1]$	$[\text{NP}, 3, 4, \alpha_1, 22]$	
$[\text{VP}, 2, 4, \alpha_1, 2]$		
$[\text{S}, 1, 4, \alpha_1, \epsilon]$		

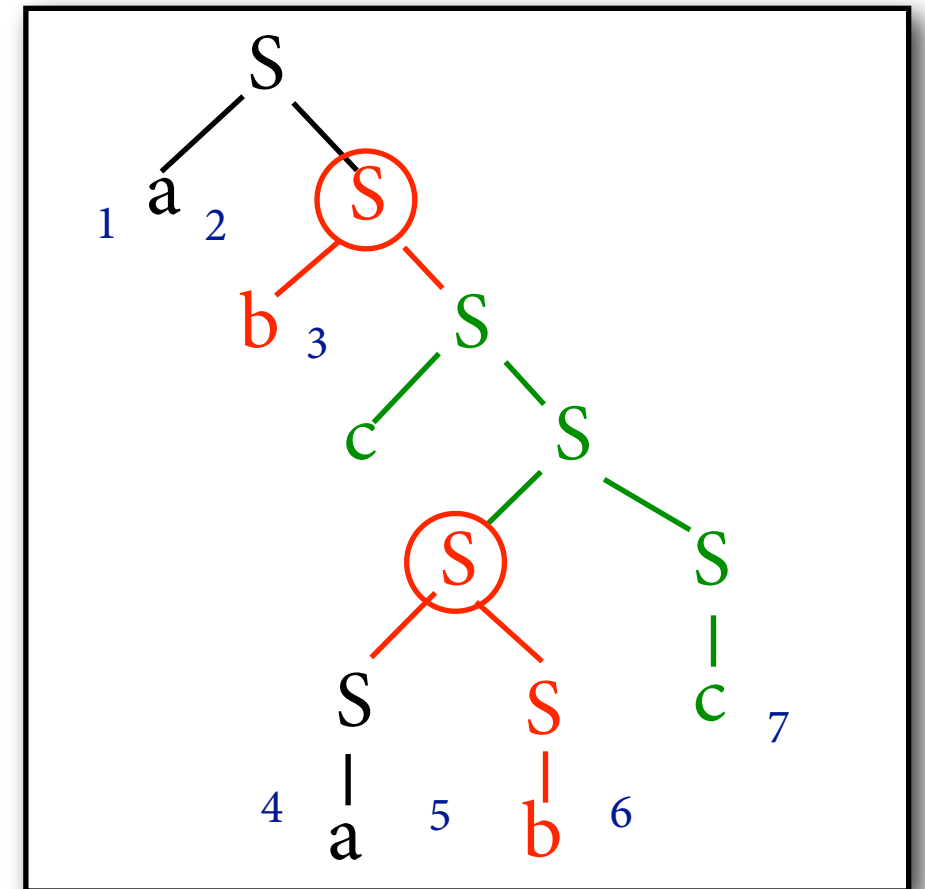
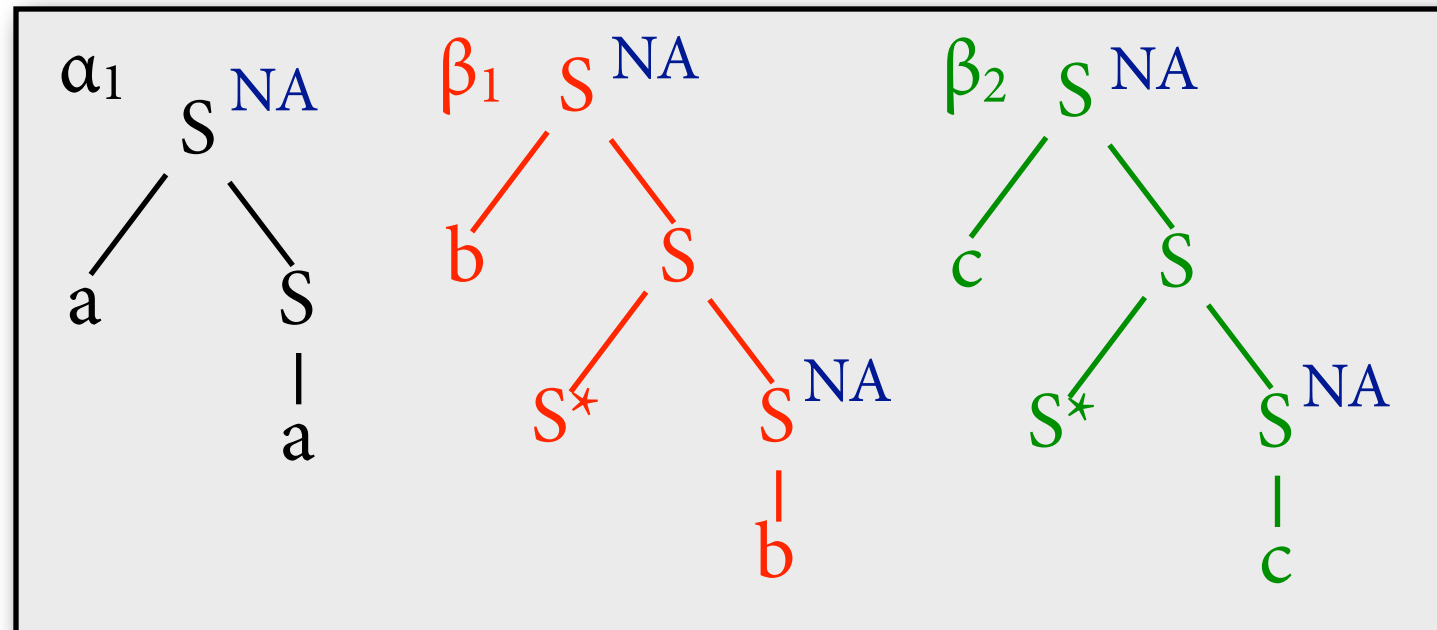


# Jetzt zu TAG



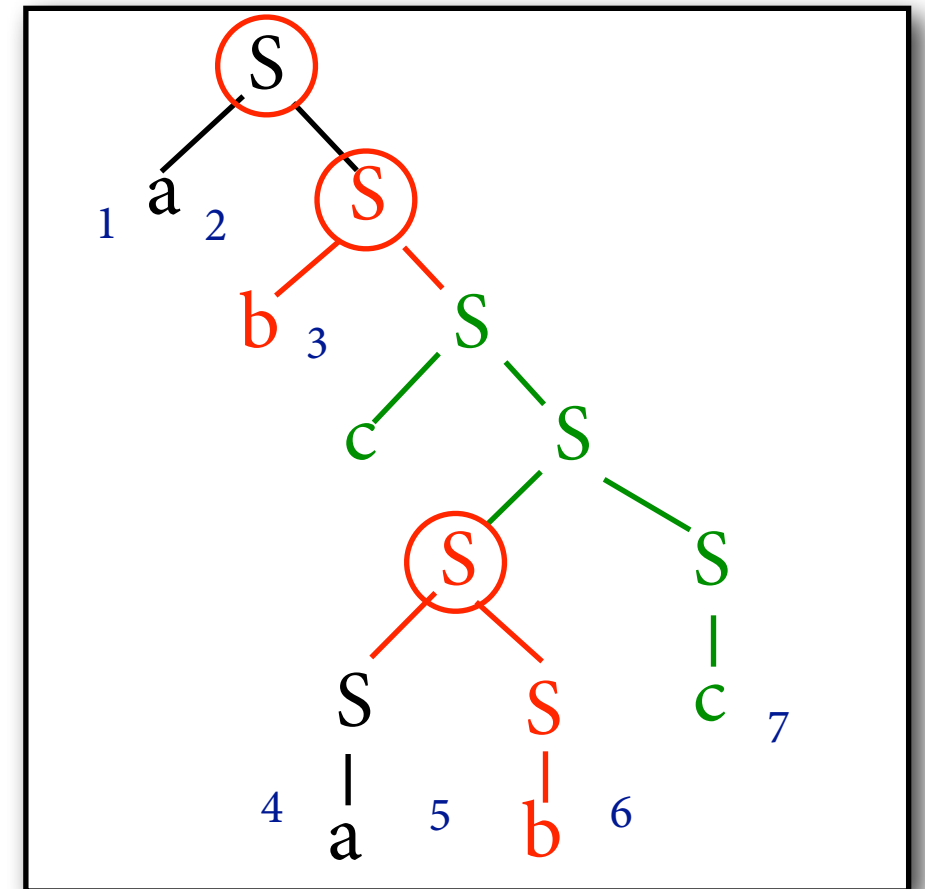
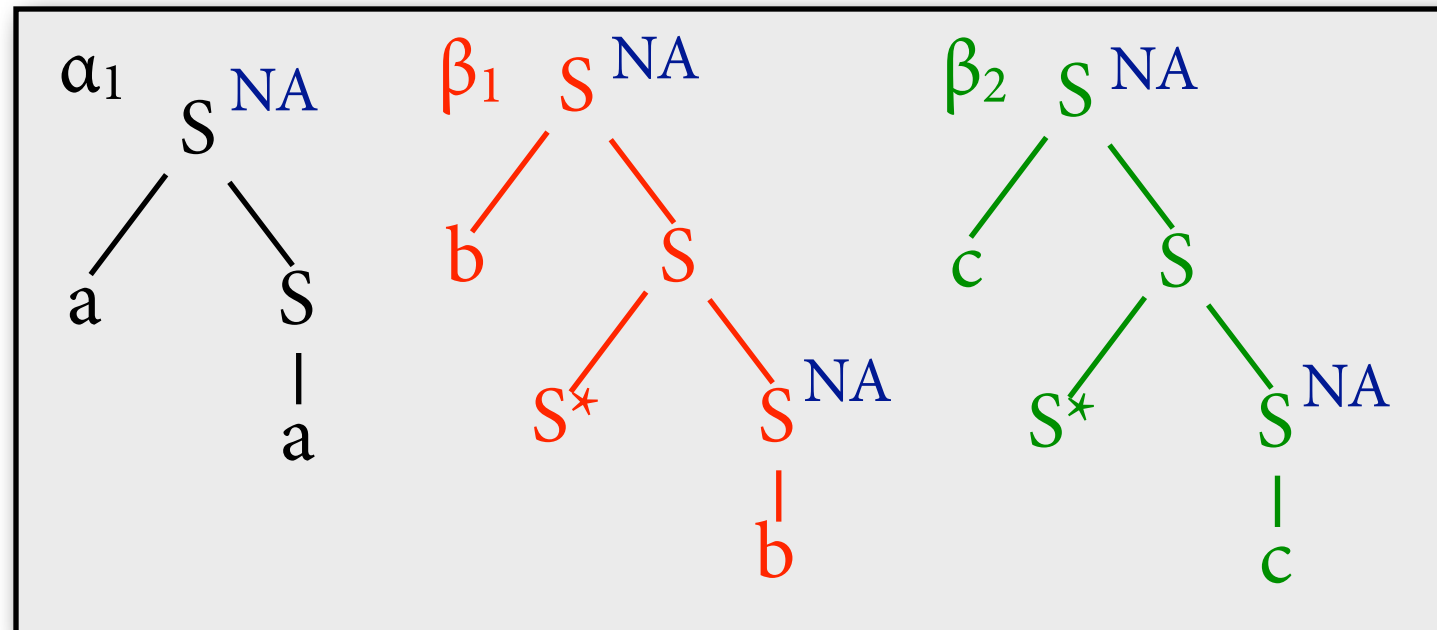
1. Versuch:  $\frac{[S, 4, 5, \alpha_1, 2] \quad [S, 5, 6, \beta_1, 22]}{[S, 4, 6, \beta_1, 2]}$

# Jetzt zu TAG



1. Versuch: 
$$\frac{[S, 4, 5, \alpha_1, 2] \quad [S, 5, 6, \beta_1, 22]}{[S, 4, 6, \beta_1, 2]}$$
$$\vdots$$
$$\underline{[S, 2, 7, \beta_1, \epsilon]}$$

# Jetzt zu TAG



1. Versuch:

$$\frac{[S, 4, 5, \alpha_1, 2] \quad [S, 5, 6, \beta_1, 22]}{[S, 4, 6, \beta_1, 2]}$$

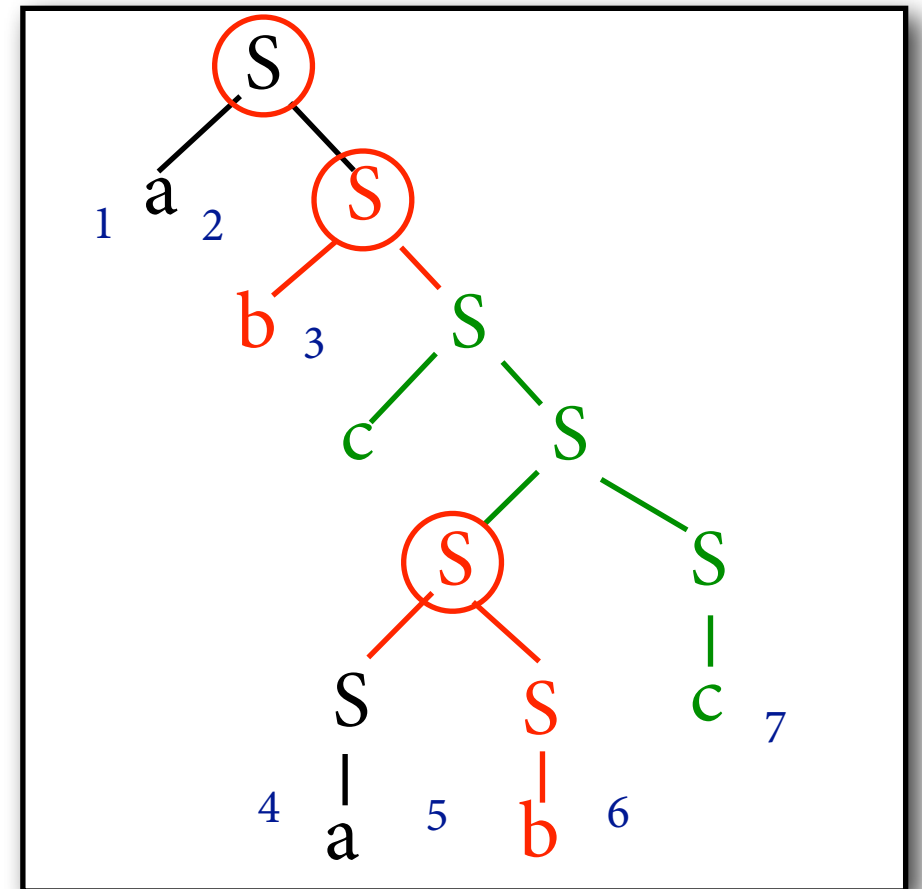
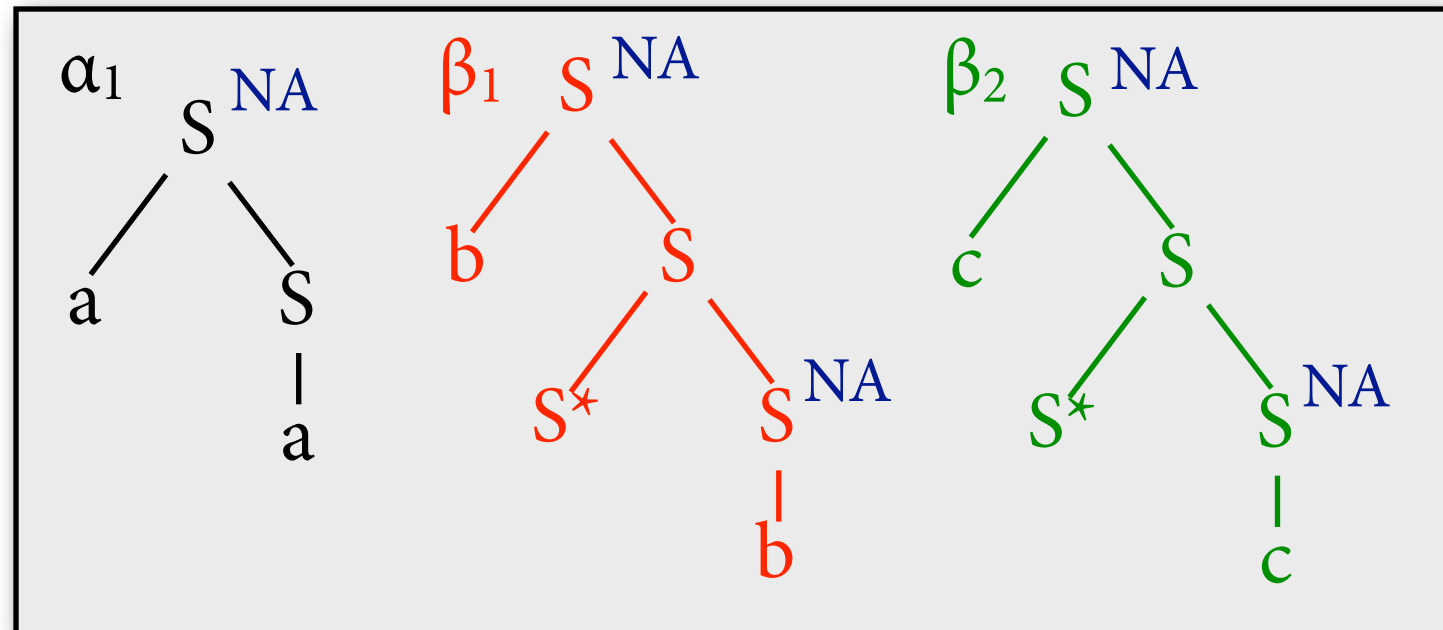
$$\vdots$$

$$\frac{[S, 2, 7, \beta_1, \epsilon]}{[S, 2, 7, \alpha_1, 2]}$$

$$\vdots$$

$$\frac{[a, 1, 2, \alpha_1, 1] \quad [S, 2, 7, \alpha_1, 2]}{[S, 1, 7, \alpha_1, \epsilon]}$$

# Jetzt zu TAG



1. Versuch:

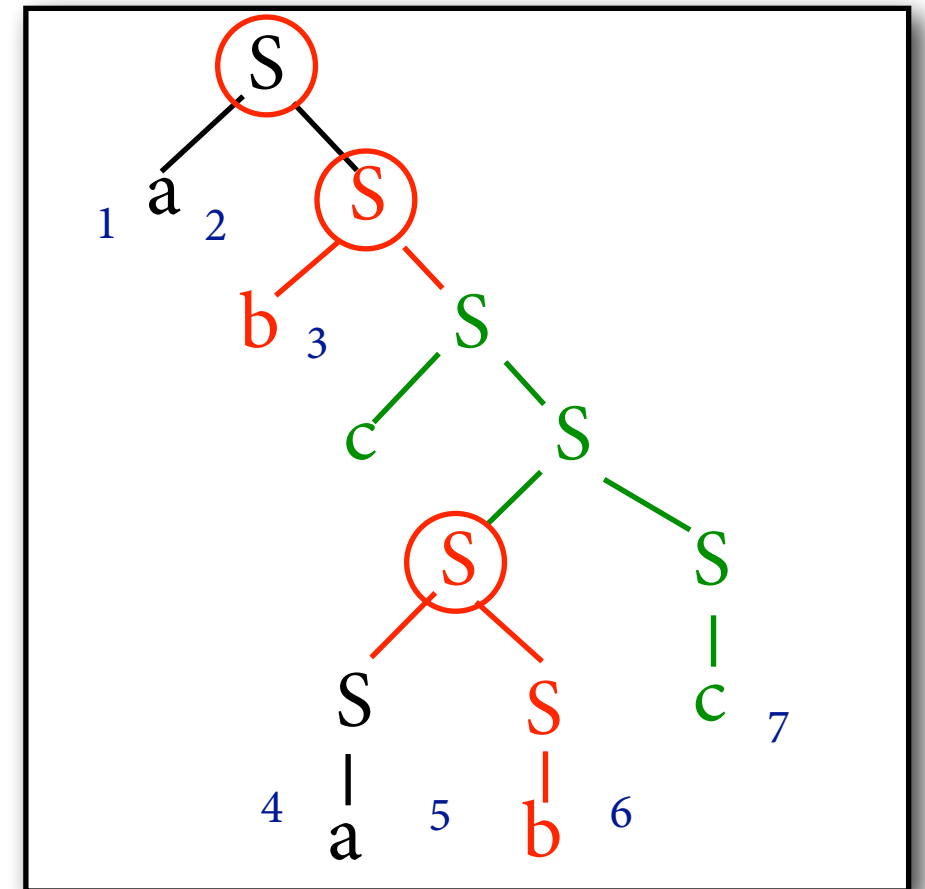
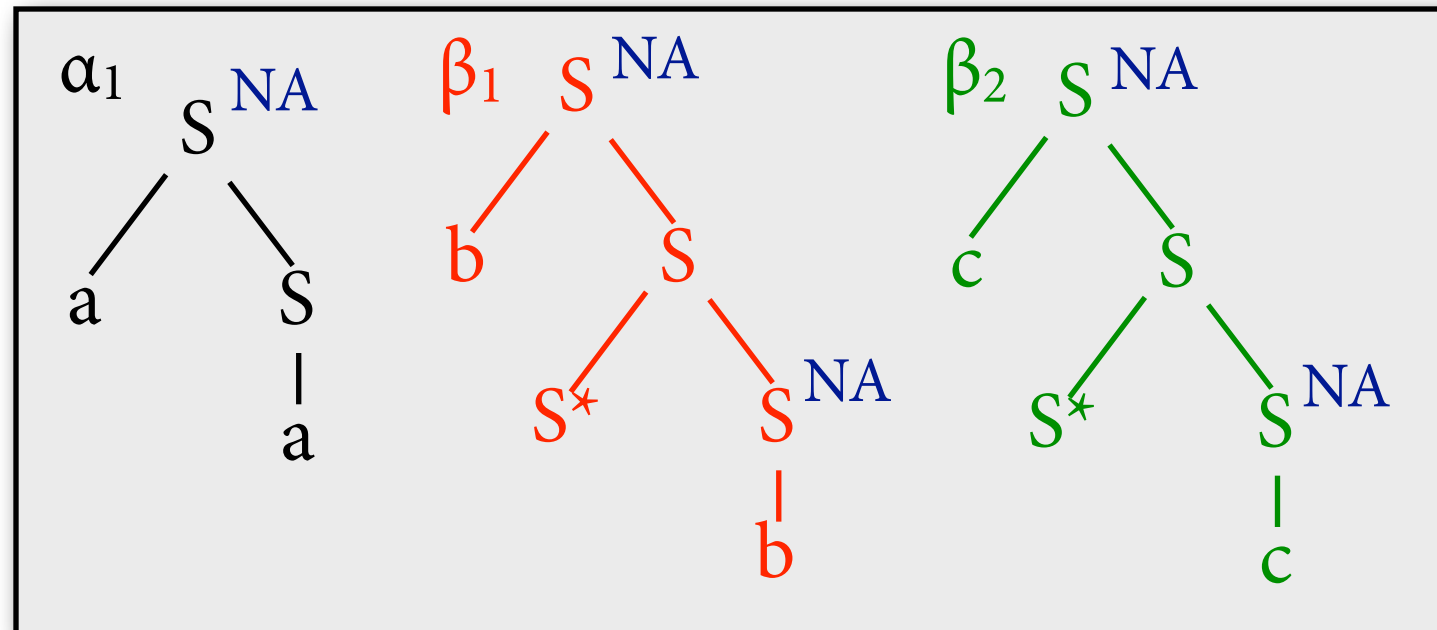
$$\frac{[S, 4, 5, \alpha_1, 2] \quad [S, 5, 6, \beta_1, 22]}{[S, 4, 6, \beta_1, 2]}$$

$$\vdots$$

$$\frac{[S, 2, 7, \beta_1, \epsilon]}{\vdots \text{ ???}}$$

$$\frac{[a, 1, 2, \alpha_1, 1] \quad [S, 2, 7, \alpha_1, 2]}{[S, 1, 7, \alpha_1, \epsilon]}$$

# Jetzt zu TAG



1. Versuch:

$$\begin{array}{c}
 \frac{[S, 4, 5, \alpha_1, 2] \quad [S, 5, 6, \beta_1, 22]}{[S, 4, 6, \beta_1, 2]} \\
 \vdots \\
 \frac{[S, 2, 7, \beta_1, \epsilon]}{[S, 2, 7, \alpha_1, 2]} \\
 \vdots \text{ ??? } \\
 \frac{[a, 1, 2, \alpha_1, 1] \quad [S, 2, 7, \alpha_1, 2]}{[S, 1, 7, \alpha_1, \epsilon]}
 \end{array}$$

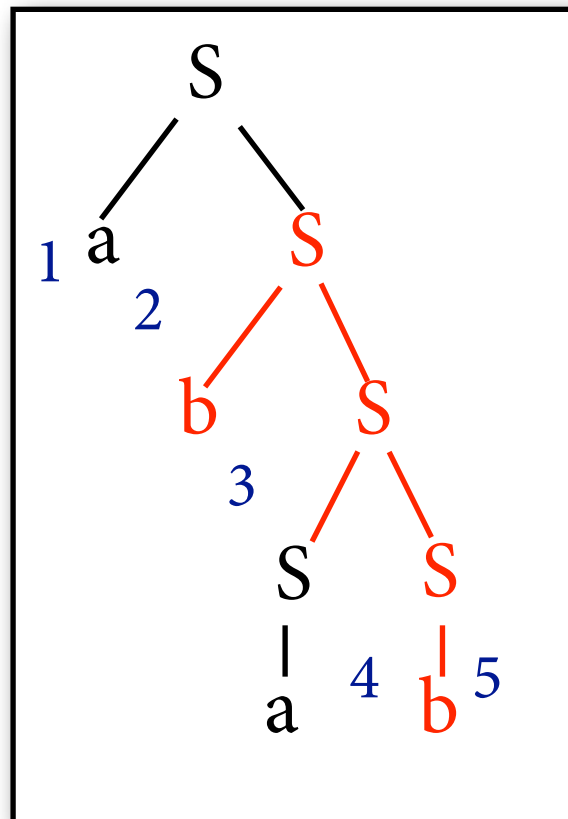
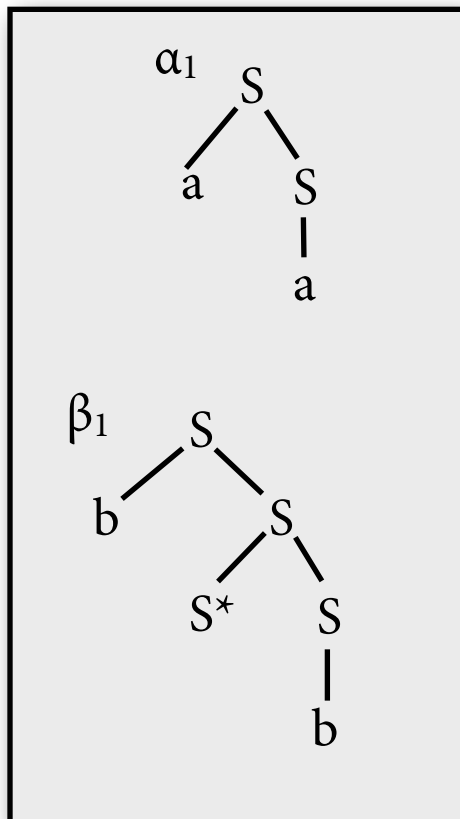
# Problem

- Bei einer Adjunktion von  $\beta$  in  $\alpha$  verarbeitet Parser erst mal  $\beta$  und “vergisst”, dass er noch den Rest von  $\alpha$  verarbeiten muss.
- Eine Lösung: Items enthalten Stack von unverarbeiteten Baumteilen.
- Führt aber zu exponentieller Laufzeit.
  - ▶ Zeige ich Ihnen in einer analogen Situation bei CCG.



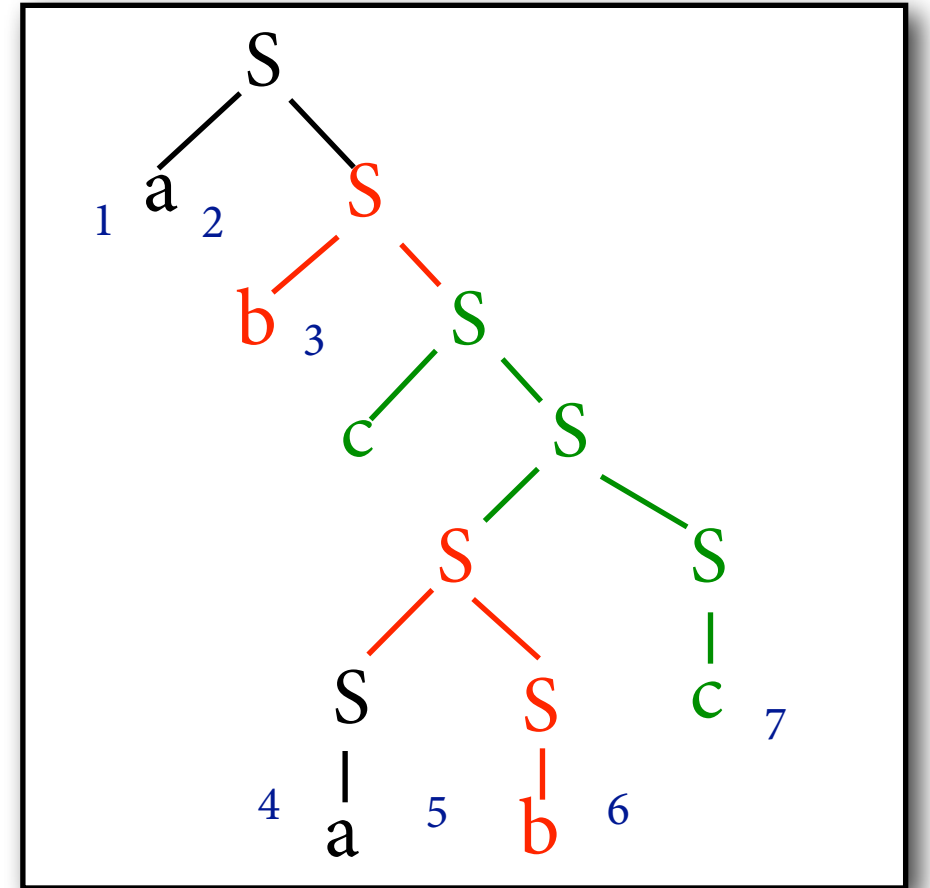
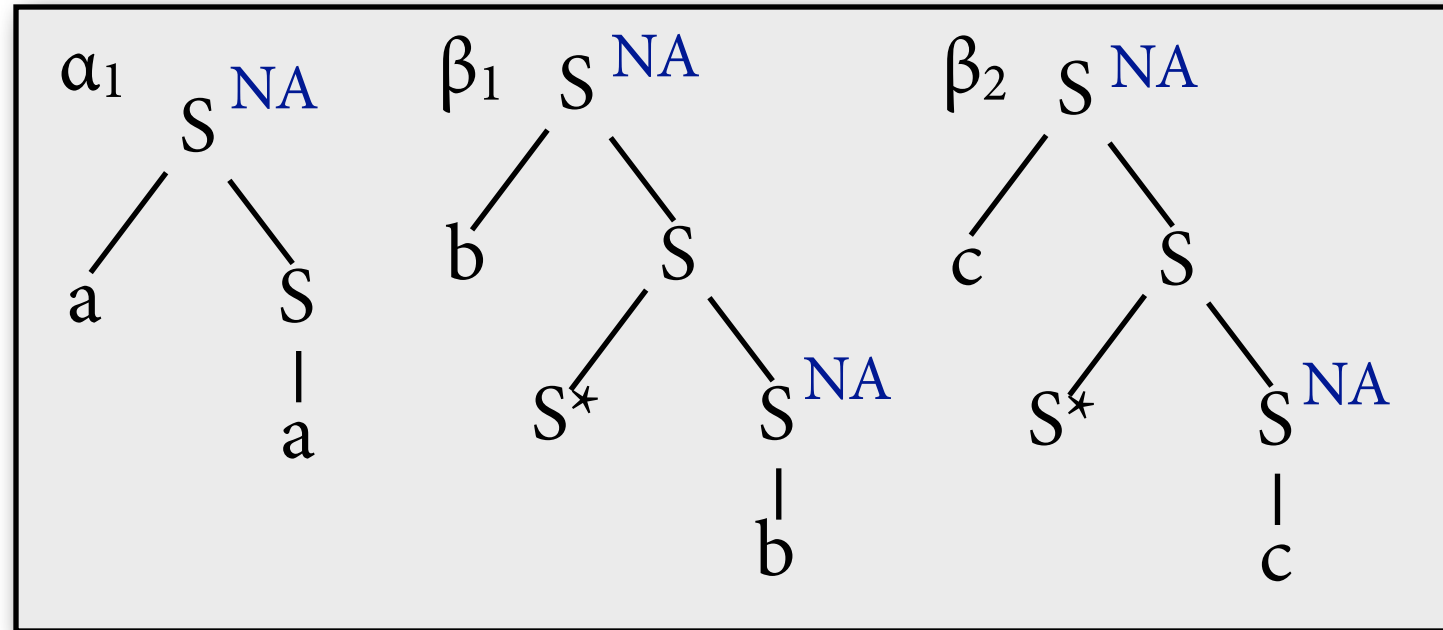
# CKY-Parser für TAG

- Zwei Typen von Items:
  - ▶  $[A, i, k, \alpha, \pi]$ : Abgeleiteter Baum unter Knoten  $\pi$  in  $\alpha$  deckt vollständigen String von  $i$  bis  $k$  ab (wie in TSG).
  - ▶  $\langle A, i-j, k-l, \beta, \pi \rangle$ : Abgeleiteter Baum unter  $\pi$  in  $\beta$  deckt String von  $i-j$  sowie *String von  $k-l$*  ab; dazwischen ist "Lücke".



$$\frac{\langle S, 2-3, 4-5, \beta_1, \epsilon \rangle \quad [S, 3, 4, \alpha_1, 2]}{[S, 2, 5, \alpha_1, 2]}$$

# Beispiel



$$\begin{array}{c}
 \frac{[S, 1-2, \alpha_1, 1] \quad \frac{[S, 2-3, \beta_1, 1] \quad \frac{\frac{\langle S, 3-4, 6-7, \beta_2, \epsilon \rangle \quad \langle S, 4-4, 5-7, \beta_1, 2 \rangle}{\text{conc12}} \quad \langle S, 2-4, 5-7, \beta_1, \epsilon \rangle}{\text{conc11}} \quad [S, 2-7, \alpha_1, 2] \quad [S, 4-5, \alpha_1, 2]}{[S, 1-7, \alpha_1, \epsilon]} \quad \frac{\frac{\frac{\langle S, 3-4, 6-7, \beta_2, \epsilon \rangle \quad \langle S, 4-4, 5-6, \beta_1, 2 \rangle}{\text{wrap22}} \quad \langle S, 4-4, 5-5, \beta_1, 21 \rangle \quad [S, 5-6, \beta_1, 22]}{\text{conc21}}}{[S, 1-7, \alpha_1, \epsilon]}
 \end{array}$$

# TAG-Parser: Regeln

$$\frac{\langle A, i_1, i_2, i_3, i_4, \beta, \epsilon \rangle \quad [A, i_2, i_3, \alpha, \pi]}{[A, i_1, i_4, \alpha, \pi]} \quad \text{wrap21}$$

$$\frac{\text{Knoten } \pi \text{ von } \alpha \text{ ist Wort } w_i}{[w_i, i, i + 1, \alpha, \pi]} \quad \text{lex}$$

$$\frac{\text{Knoten } \pi \text{ von } \beta \text{ ist Fußknoten } A^*, \text{ und } i < k}{\langle A, i, i, k, k, \beta, \pi \rangle} \quad \text{foot}$$

NB: “foot” leitet Items für beliebige  $i, k$  ab.

# TAG-Parser: Regeln

$$\frac{\langle B, i_1, i_2, i_3, i_4, \beta, \pi 1 \rangle \quad [C, i_4, i_5, \beta, \pi 2]}{\langle A, i_1, i_2, i_3, i_5, \beta, \pi \rangle} \quad \text{conc21}$$

$$\frac{[B, i_1, i_2, \beta, \pi 1] \quad \langle C, i_2, i_3, i_4, i_5, \beta, \pi 2 \rangle}{\langle A, i_1, i_3, i_4, i_5, \beta, \pi \rangle} \quad \text{conc12}$$

$$\frac{\langle A, i_1, i_2, i_5, i_6, \beta_1, \epsilon \rangle \quad \langle A, i_2, i_3, i_4, i_5, \beta_2, \pi \rangle}{\langle A, i_1, i_3, i_4, i_6, \beta_2, \pi \rangle} \quad \text{wrap22}$$

# Diskussion

- Laufzeit: Teuerste Regel ist wrap22.  
Sechs unabhängige Stringpositionen, also  $O(n^6)$ .
- Kann Algorithmus auf TAG-Grammatiken mit mehr als zwei Kindern pro Knoten erweitern (analog Earley-Parser für kfGs).
- Algorithmus erlaubt mehrfache Adjunktion am gleichen Knoten. Kann man verbieten: Item merkt sich, ob schon adjungiert wurde.

# Zusammenfassung

- Expressivität von TAG:  
kann COPY, COUNT(4), nicht mehr COUNT(5)
- Parsing von TAG: mit CKY-artigem Parser in  $O(n^6)$ .