Effizientes Parsing und Generieren mit Baumadjunktions-grammatiken (TAG)

CYK-basiertes Parsing

- Der Algorithmus von K. Vijay-Shanker

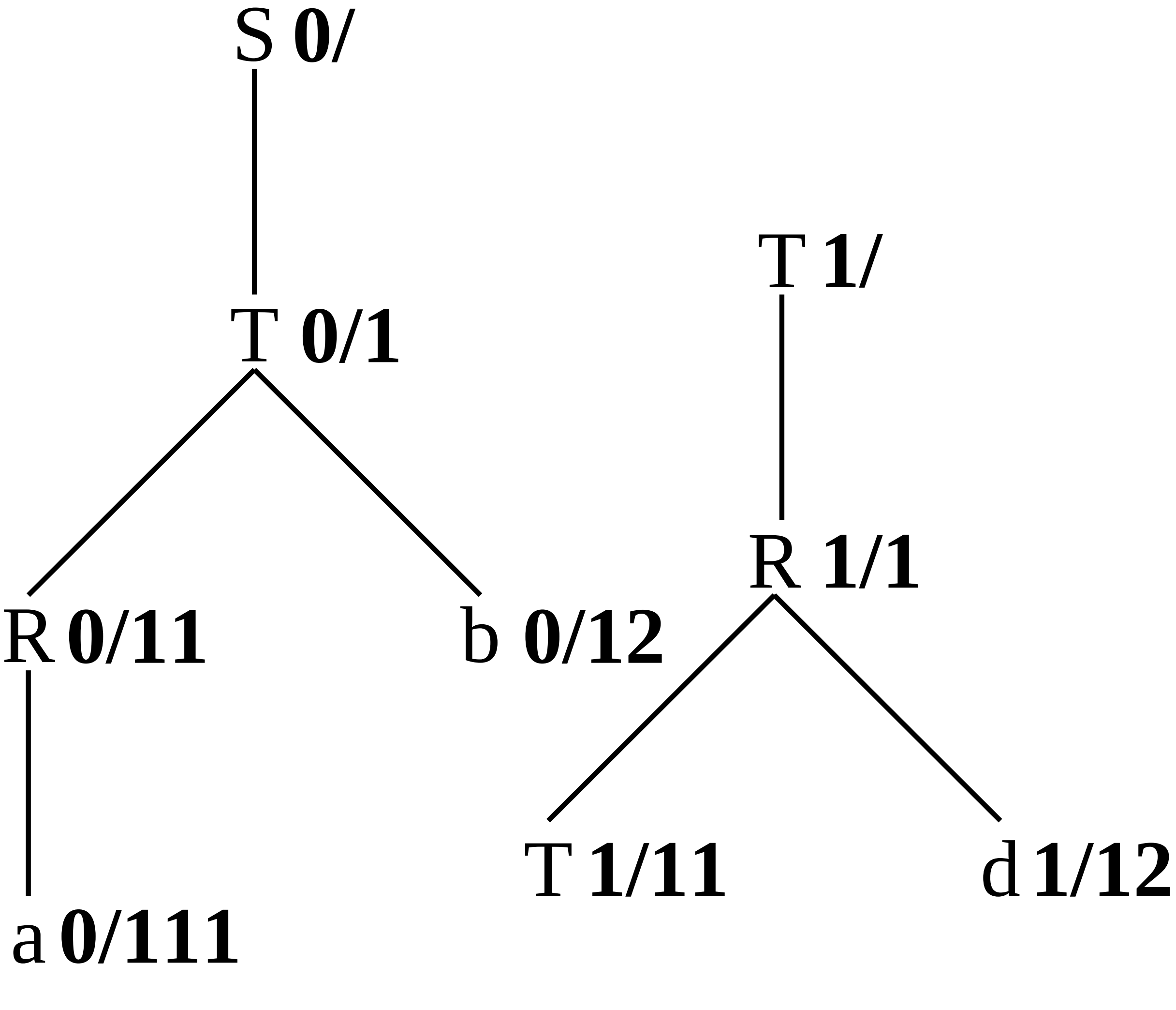
- Ein alternativer Ansatz von K. Harbusch

Definition von Knotennummern

Die elementaren Bäume seien durchnumeriert.

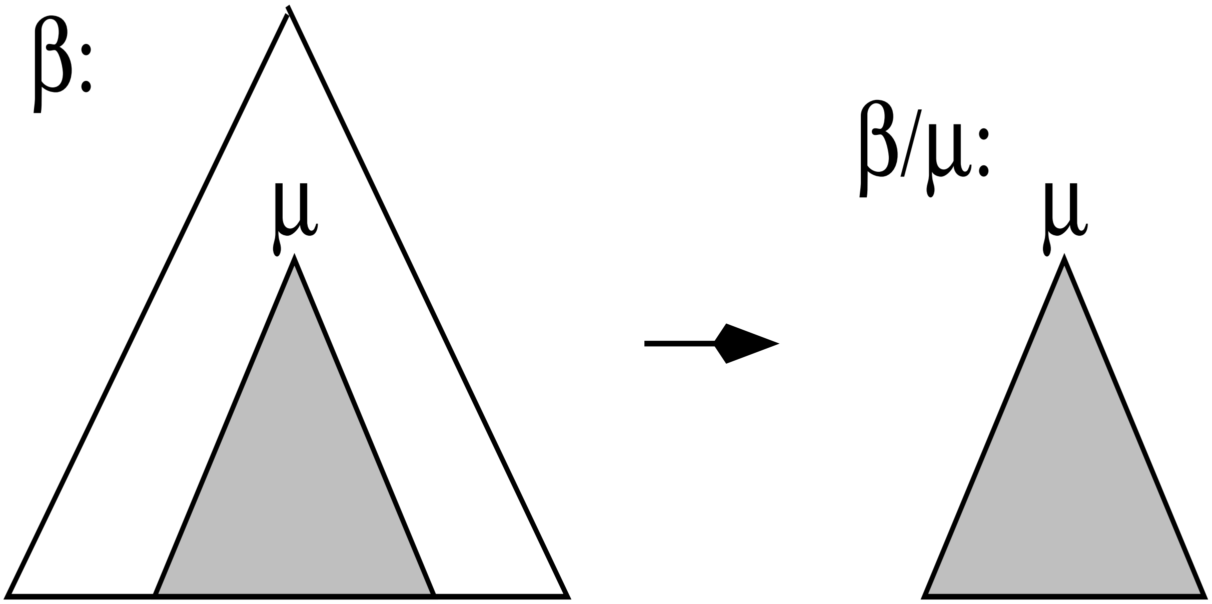
Die Knotennummer sei dann:
$Baumnummer \cdot / \cdot Knotenadresse$**.**

Bsp.:



Definition der Menge $\mathcal{D}(\mu)$

Sei β ein elementarer Baum und μ ein Knoten darin:



$\mathcal{D}(\mu)$ **sei wie folgt definiert:**

- **Der von μ dominierte Unterbaum in β (bez. mit β/μ) ist in $\mathcal{D}(\mu)$.**

- **Ist γ in $\mathcal{D}(\mu)$, dann auch jeder von γ mittels Adjunktion abgeleiteter Baum.**

Bedingungen an die TAG beim Algorithmus von Vijay-Shanker

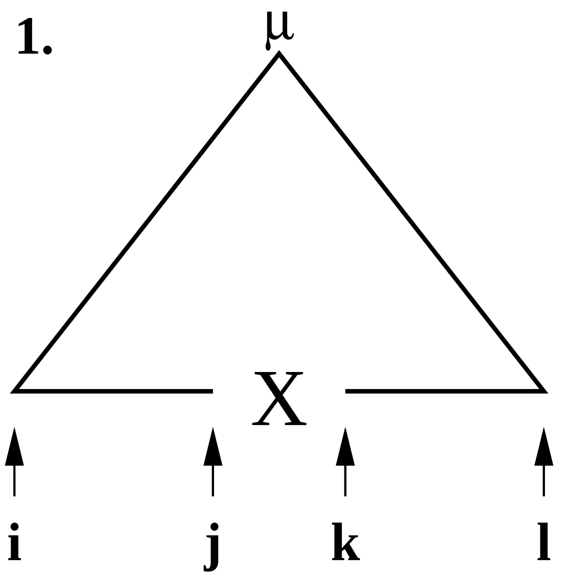
- Die Grammtik ist in Zweiform, d.h. jeder Knoten hat höchstens 2 Söhne.
- Jeder Baum produziert mindestens ein Terminalsymbol.
- Es sind keine Knoten mit der Beschriftung ϵ erlaubt.

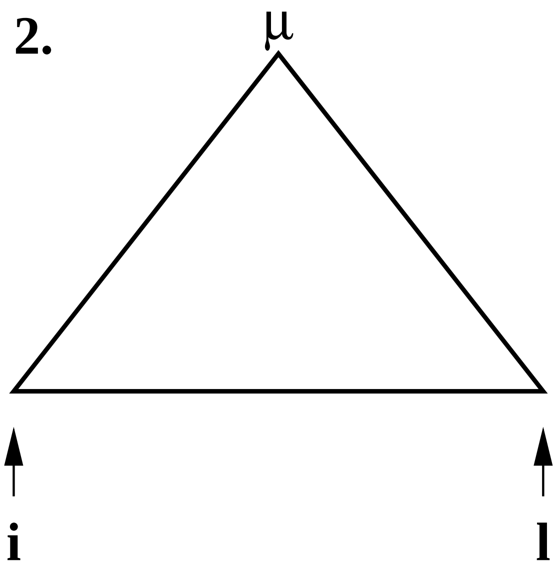
Bedeutung des Feldes $A[i,j,k,l]$ beim Parsen der Eingabe $a_1 \cdots a_n$:

Sei μ die Nummer eines Knoten in einem Elementarbaum, dessen Wurzel die Beschriftung X trägt:

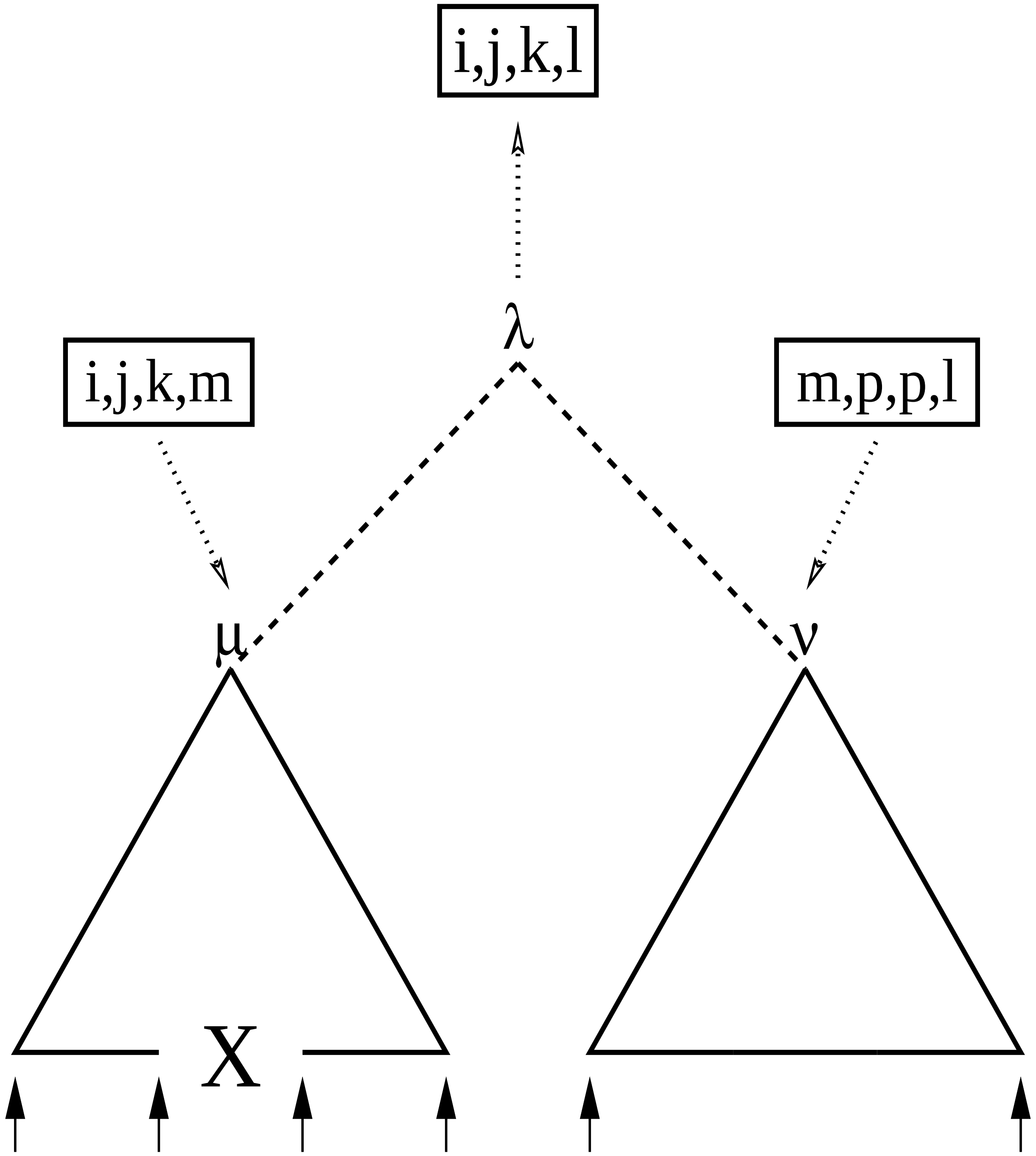
1. Für i,j,k,l mit $0 \leq i \leq j < k \leq l \leq n$ gilt:
   $\mu \in A[i,j,k,l] \Longleftrightarrow \exists\gamma \in \mathcal{D}(\mu)$ mit Blattwort $a_{i+1} \cdots a_j \; X \; a_{k+1} \cdots a_l$

2. Für i,l mit $0 \leq i < l \leq n$ gilt:
   $\exists \; i \leq j \leq l : \; \mu \in A[i,j,j,l] \Longleftrightarrow \exists\gamma \in \mathcal{D}(\mu)$ mit Blattwort $a_{i+1} \cdots a_l$

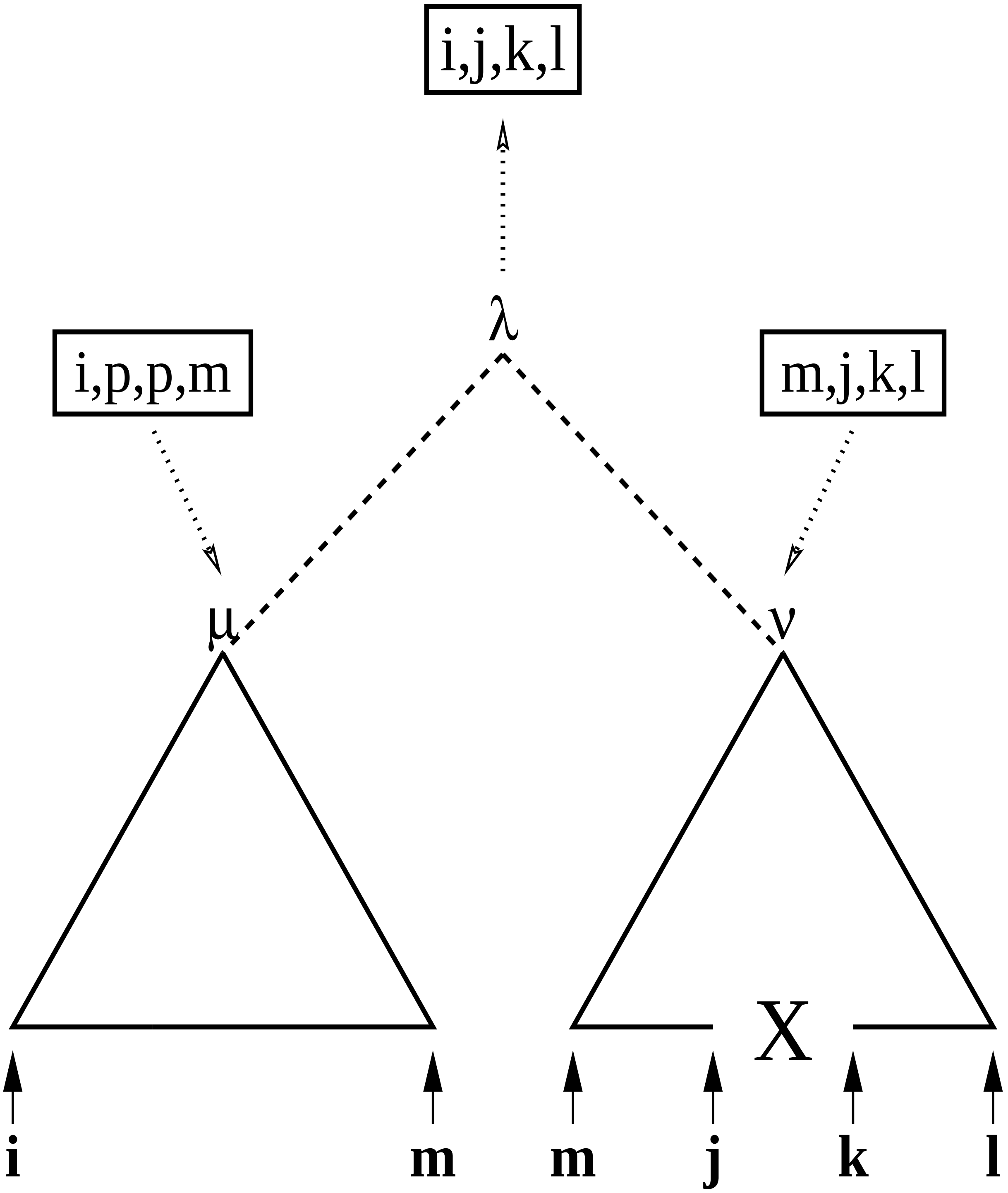




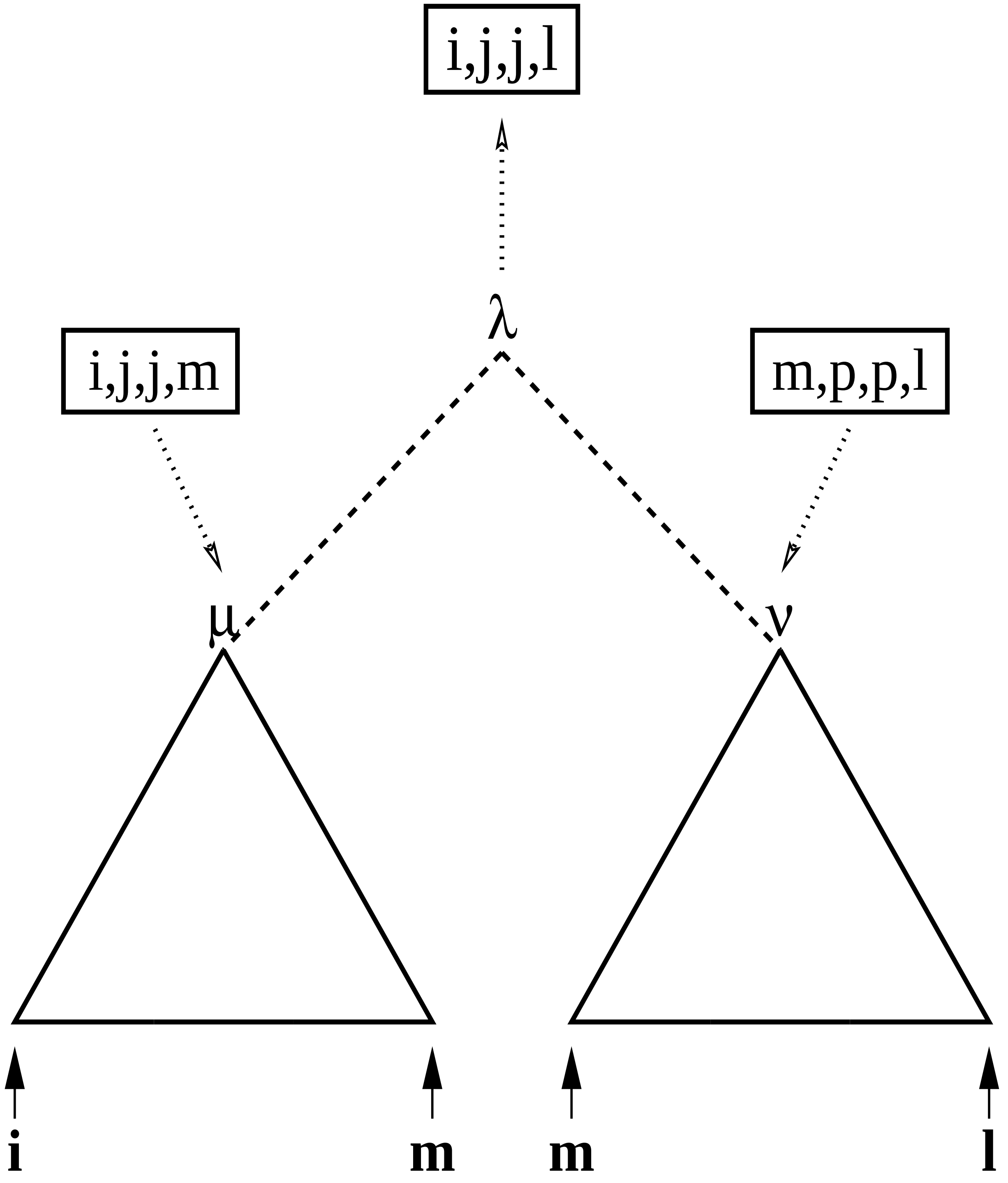
Case 1



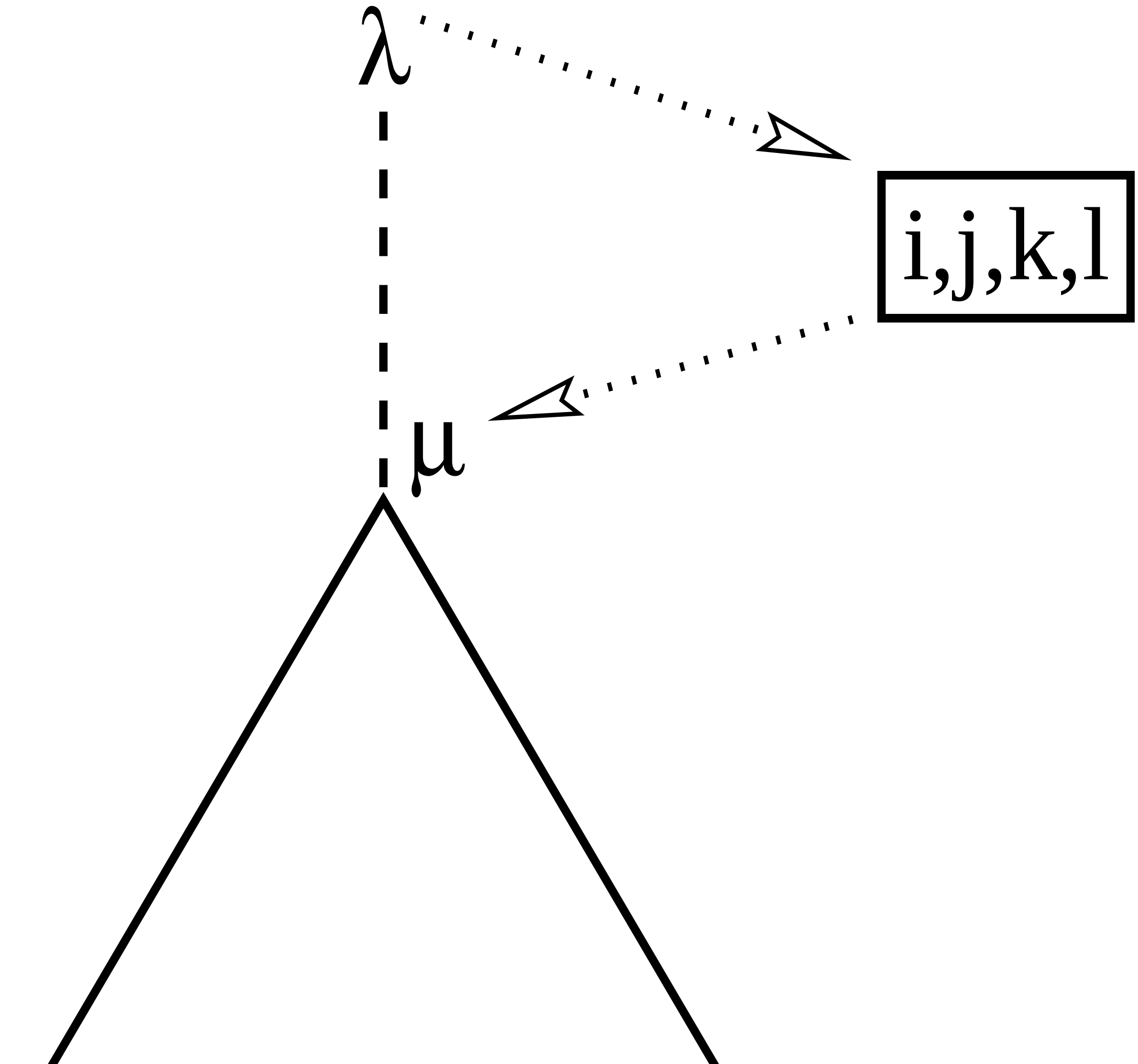
Case 2



Case 3

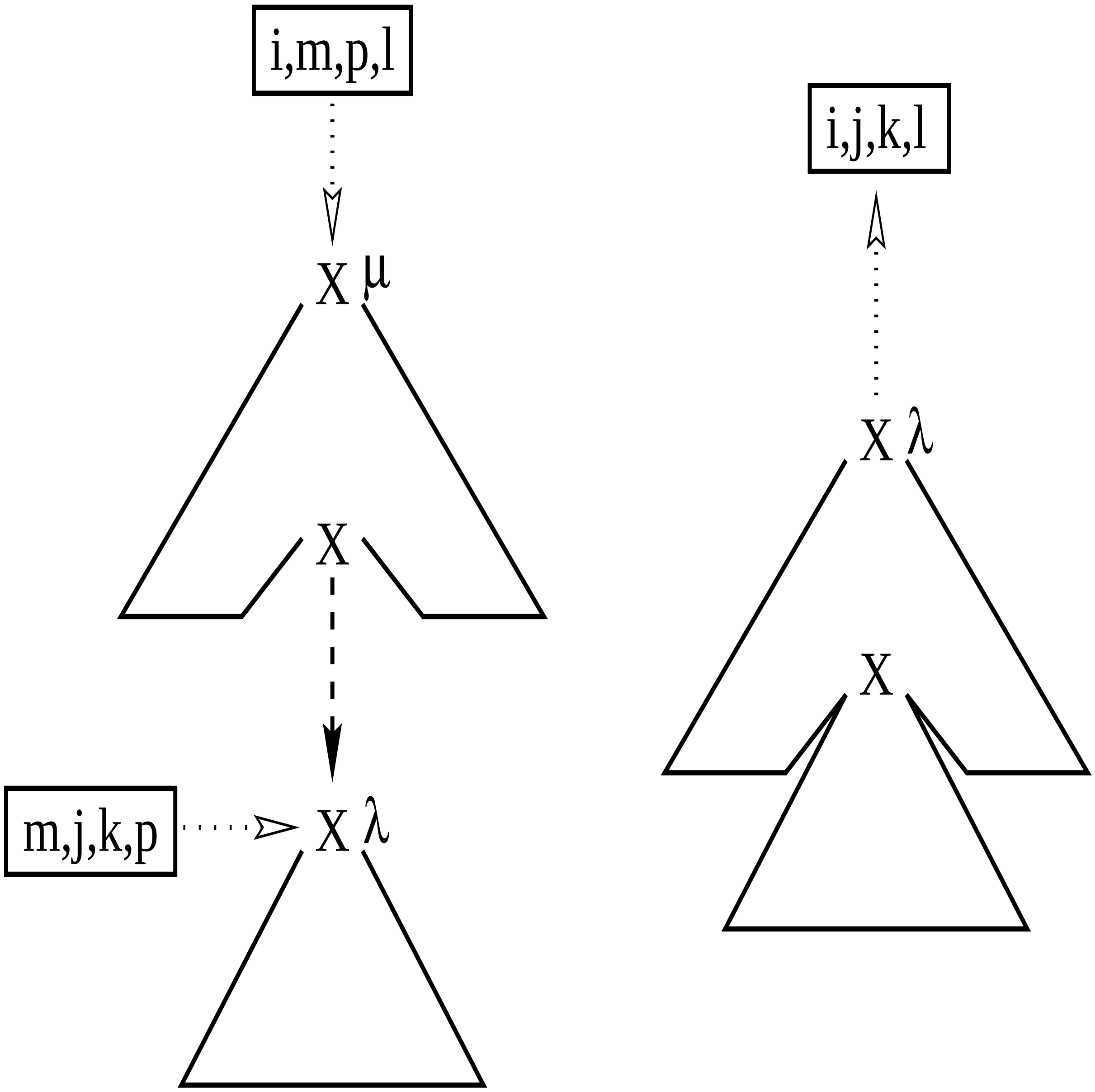


Case 4



Diese Konstruktion muß eventuell mehrmals ausgeführt werden.

Case 5 (Adjunktion)



Es wird nicht die Knotennumer μ, sondern die Knotennummer λ gespeichert.

Initialisierung des Algorithmus bei der Eingabe $a_1 \ldots a_n$

(1) for i=0 to n-1 step 1 do
(2) Schreibe alle Terminalknoten, die mit a_{i+1} beschriftet sind, in die Zelle $A[i, i+1, i+1, i+1]$
(3) for i=0 to n-1 step 1 do
(4) for j=i+1 to n step 1 do
(5) Schreibe alle Fußknoten von aux. Bäumen nach $A[i, i, j, j]$

Hauptteil des Algorithmus bei der Eingabe $a_1 \ldots a_n$

(6) for l=0 to n step 1 do
(7) for i=l to 0 step -1 do
(8) for j=i to l step 1 do
(9) for k=l to j step -1 do
(10) do Case 1
(11) do Case 2
(12) do Case 3
(13) do Case 4; do Case 5
(14) do Case 4
(15) Akzeptiere, falls die Nummer einer Wurzel eines intialen Baumes in $A[0, j, j, n]$ ist, wobei $0 \leq j \leq n$.

Bedingungen für Case 1

Das Blattwort des rechten Teilbaumes muß mindestens ein Terminalsymbol umfassen, das des linken jedoch nicht. Daraus ergibt sich die Ungleichungskette:

$$i \leq j < k \leq m < l \quad \textbf{und} \quad m \leq p \leq l$$

Oder in Pseudocode:

(C1.1) if j=k then exit;
(C1.2) for m=k to l-1 step 1 do
(C1.3)   for p=m to l step 1 do
(C1.4)   Falls der linke Sohn $\mu \in A[i,j,k,m]$ und der rechte Sohn $\nu \in A[m,p,p,l]$ ist, schreibe den Vater λ nach $A[i,j,k,l]$.

Case 2 ist symmetrisch; für Case 3 gilt Analoges.

Bedingungen für Case 5

Der Baum, der adjungiert wird, muß mindestens ein Terminalsymbol in seinem Blattwort haben. Für den (unfertigen) Teilbaum, an den adjungiert wird, gilt dies jedoch i.a. nicht.
Daraus ergibt sich die Ungleichungskette:

$$i \leq m \leq j \leq k \leq p \leq l \quad \textbf{und} \quad (i < m \vee p < l)$$

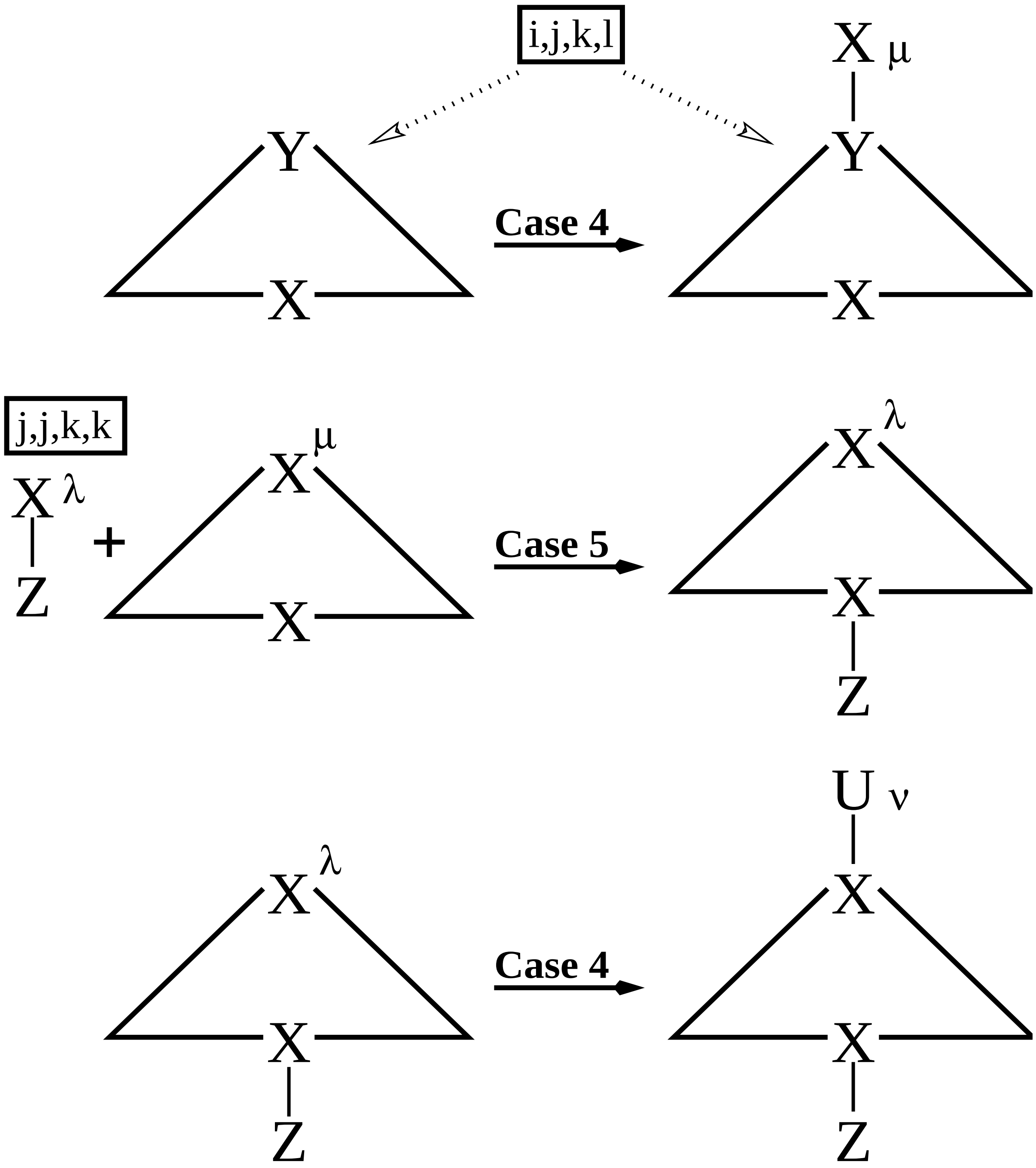
Oder in Pseudocode:
(C5.1) for m=i to j step 1 do
(C5.2) for p=k to l step 1 do
(C5.3) Ist $(i < m \vee p < l)$ **und** $\mu \in A[i, m, p, l]$ **die Wurzel eines aux. Baumes, die dieselbe Beschriftung wie** $\lambda \in A[m, j, k, p]$ **trägt, dann schreibe** λ **nach** $A[i, j, k, l]$**.**

Overlay-Folie für Case 5 mit $m = j$ und $p = k$ anfertigen:

$A[i, m, p, l]$ durch $A[i, j, k, l]$ ersetzen

$A[m, j, k, p]$ durch $A[j, j, k, k]$ ersetzen

Notwendigkeit für
Case 4 - Case 5 - Case 4



Invariante des Algorithmus

Nach dem Durchlauf der Schritte (10)-(14) mit den Werten $i_0,\ j_0,\ k_0,\ l_0$ für die zugehörigen Laufvariablen gilt:

> Die Feldelemente $A[i, j, k, l]$ mit $l \leq l_0,\ i \geq i_0,\ j \leq j_0,\ k \geq k_0$ sind korrekt berechnet.

Anmerkungen zur Korrektheit:

- Case 1-3 : greifen jeweils nur auf bereits berechnete Feldelemente zurück
- Case 4 : greift nur auf das aktuelle Feldelement zurück
- Case 5 : greift in der Regel auf bereits berechnete Feldelemente zurück, kann aber auch das aktuelle benutzen

Warum ist diese Anwendung der Fälle hinreichend ?

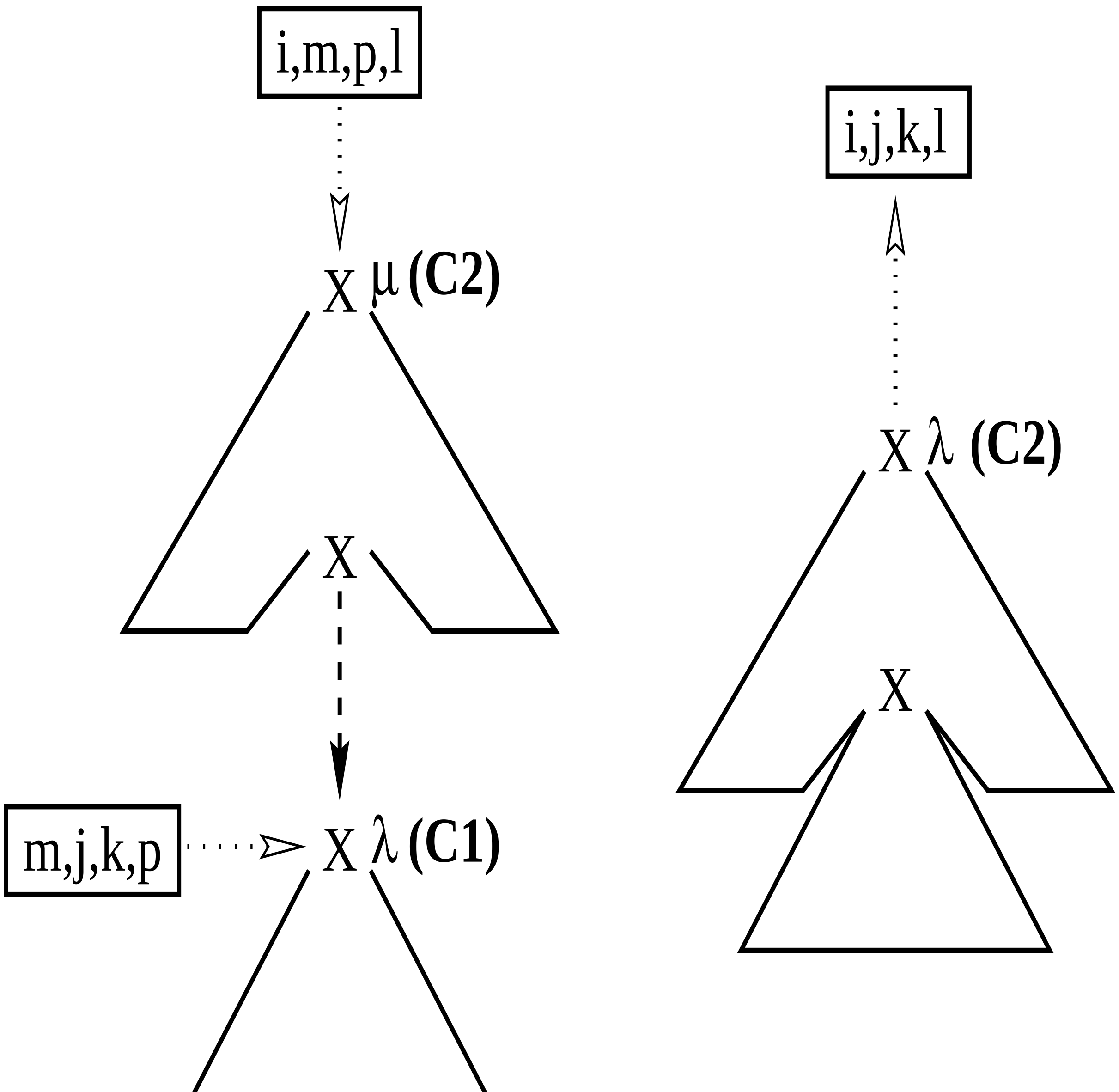
Nach dem Durchlaufen der Schritte (10)-(14) würde kein Fall mehr einen neuen Knoten hinzufügen:

- Case 1-3 : klar
- Case 4 : wurde als letztes ausgeführt
- Case 5 :
  Für $m < j$ oder $p < k$ ist das klar.

  Für $m = j$ und $p = k$ stellt man fest:
  - Durch Case 5 - Case 4 kann kein neuer Wurzelknoten zu $A[i, j, k, l]$ hinzugefügt werden, da jeder Baum mindestens ein Terminal liefert.
  - $A[j, j, k, k]$ bleibt unverändert, so daß keine neuen „Adjunktionsziele“ entstanden sein können.

Hier noch auf die Komplexität $O(n^6)$ des Algorithmus eingehen.

Case 5 mit Constraints

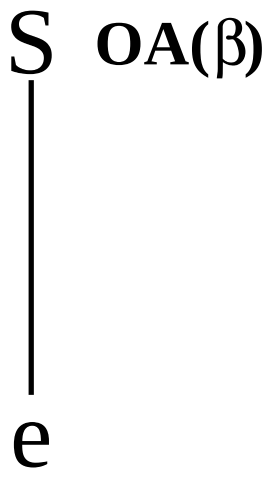


Die Constraints von λ (C1) werden über-prüft, die von μ (C2) werden gespeichert.

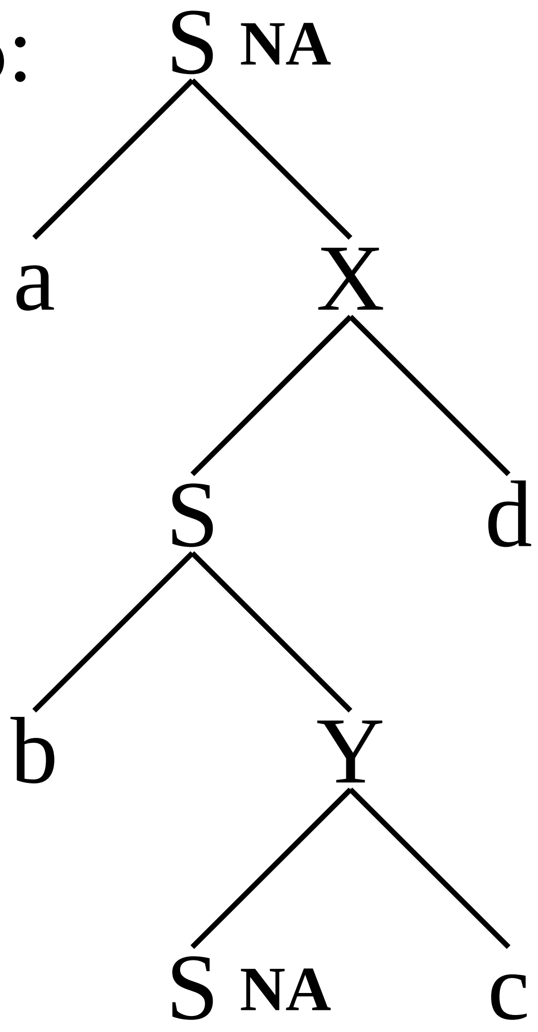
TAG zur Sprache

$$L = \{a^n b^n e c^n d^n : n \geq 1\}$$

α:



β:



- In der Besprechung Folien für Case 1, Case 2 und Case 5 wiederholen

- Auf den Bsp.-Folien den gerade betrachten Teil der Eingabe farbig markieren

Abgeleiteter Baum zu der Eingabe $aabbeccdd$

Initialisierung bei der Eingabe

$aabbeccdd$

Terminale: a

$\boxed{0,1,1,1}$ $\boxed{1,2,2,2}$

...

d

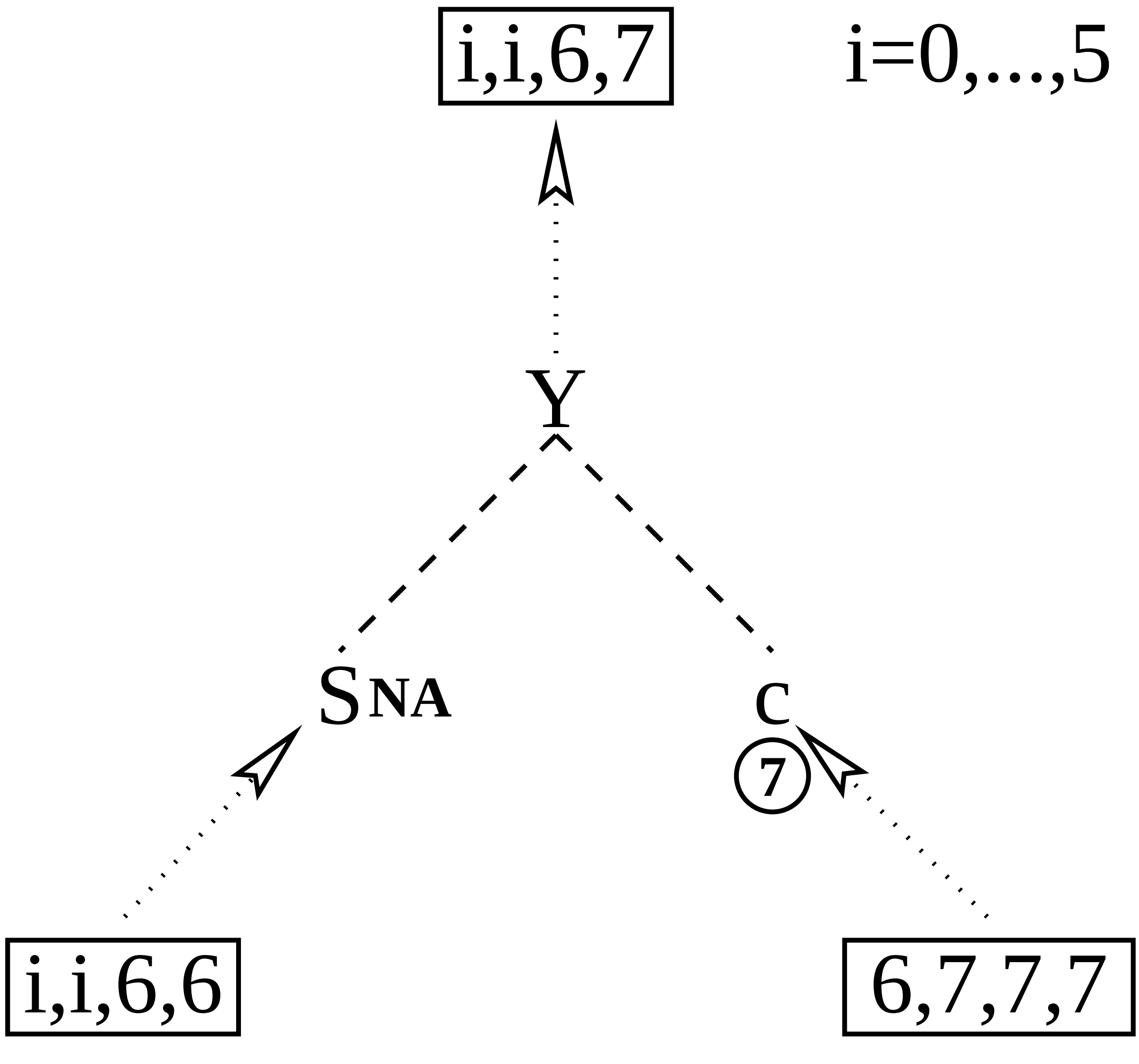
$\boxed{7,8,8,8}$ $\boxed{8,9,9,9}$

Fußknoten: S_{NA}

$\boxed{0,0,1,1}$ ... $\boxed{8,8,9,9}$

Anwendung von Case 1

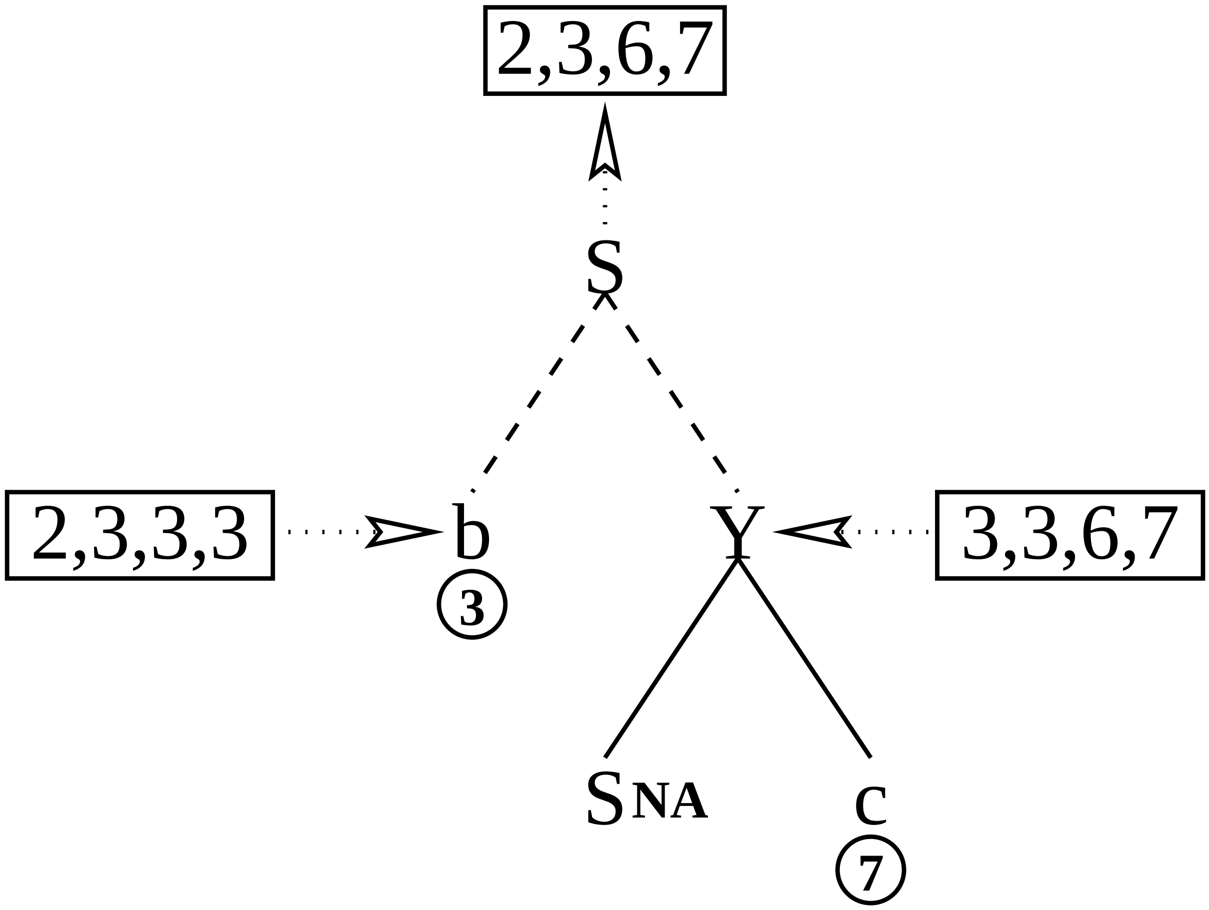
Eingabe $aabbeccdd$



Analog wird $A[i,i,5,6]$ mit $i = 0,\ldots,4$ für das c an 6. Stelle der Eingabe aufgebaut.

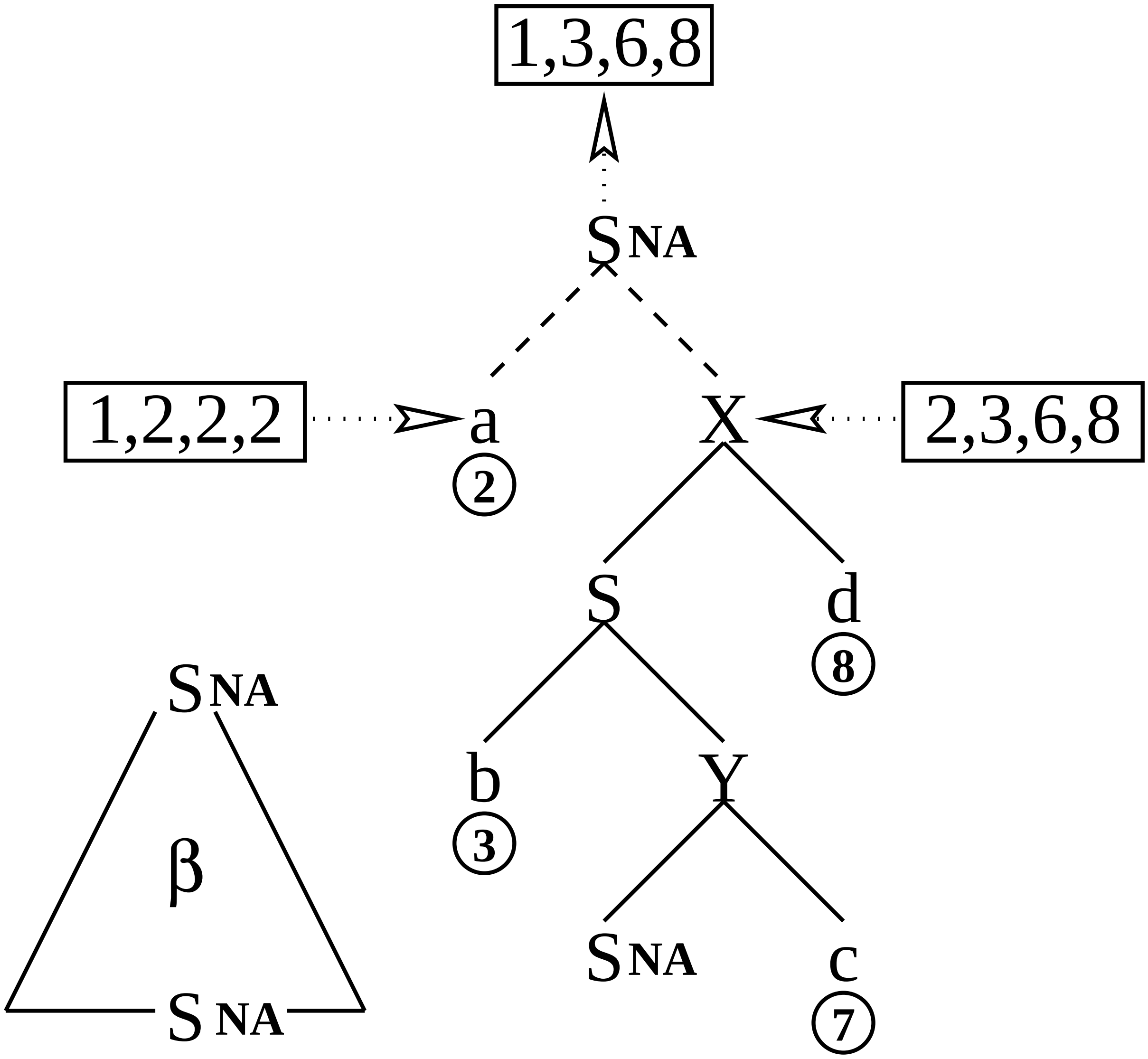
Anwendung von Case 2

Eingabe $aabbeccdd$



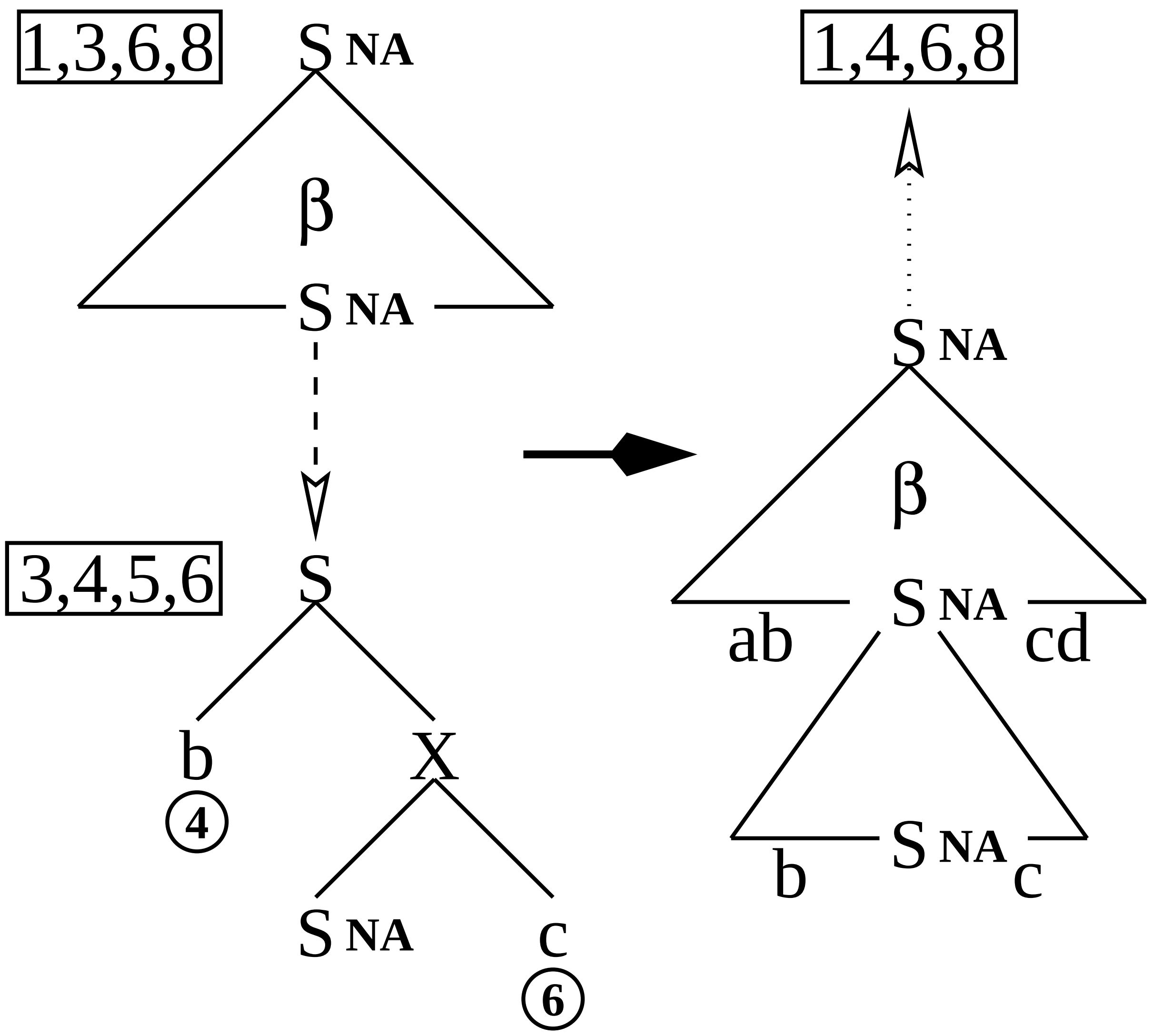
Analog wird $A[3,4,5,6]$ für das b an 4. und das c an 6. Stelle der Eingabe aufgebaut.

Der erste vollständige elementare Baum Eingabe $aabbeccdd$



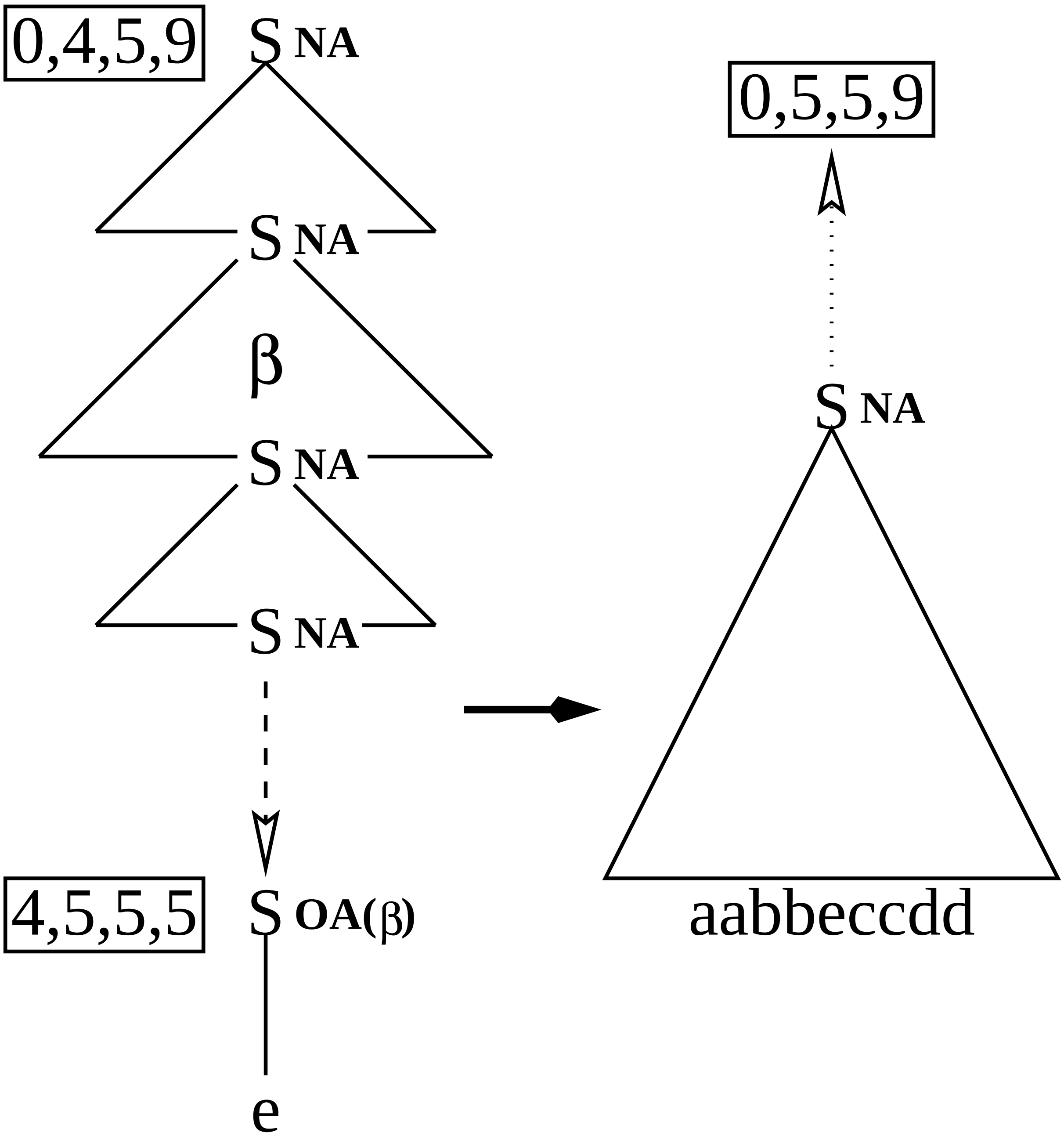
Anwendung von Case 5

Eingabe $aabbeccdd$



Der letzte Schritt

Eingabe *aabbeccdd*



Erweiterung auf ϵ - Knoten

Änderungen am Algorithmus:

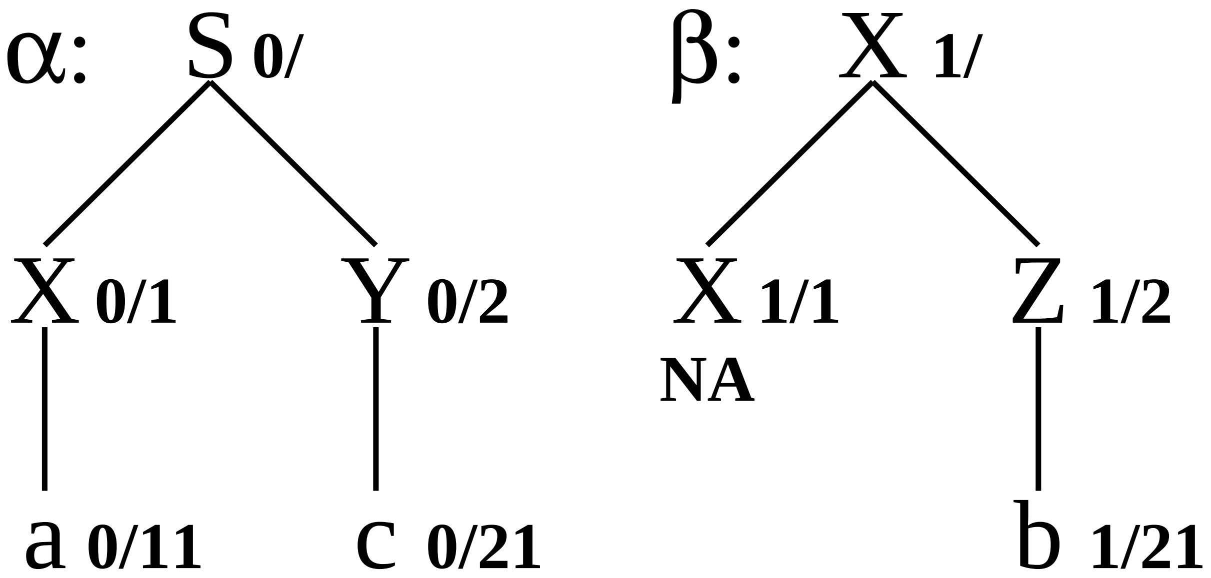
- **Initialisierung:**
  for i=0 to n step 1 do
  Schreibe alle ϵ-Knoten nach $A[i,i,i,i]$
- **Case 1: Bedingung $m < l$ in $m \leq l$ umwandeln**
- **Case 2,3: analog wie Case 1**
- **Case 4,5: bleiben unverändert**

Problem: Alle Fälle greifen auf das aktuelle Feldelement zu.

Lösung:

repeat
do Case 1; . . . do Case 5;
until Keine neuen Knoten gefunden

Kontextfreier Kern
und
KFK mit Annotationen



KFK mit Annotationen:

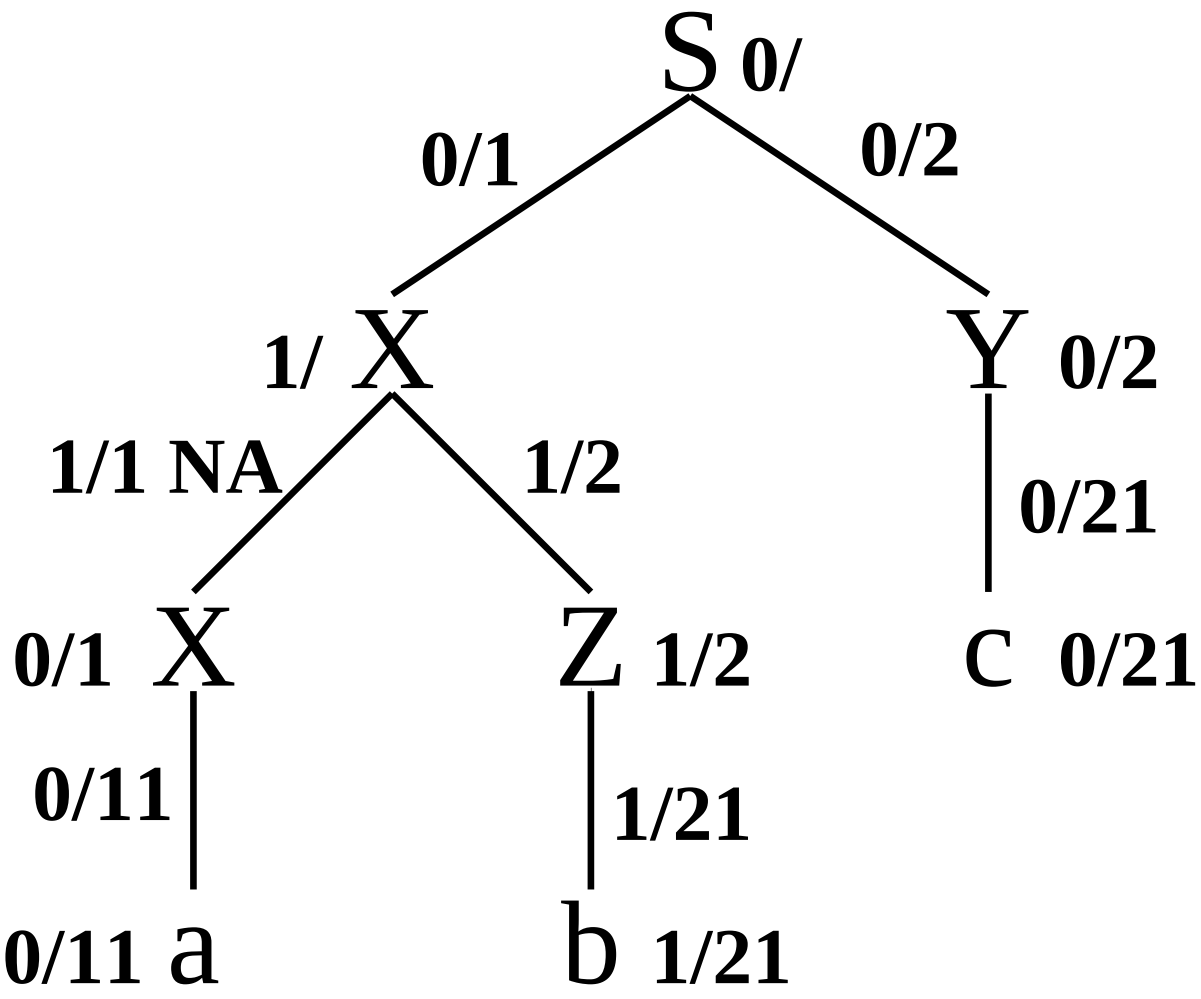
1:	**S** (0/,-)	$\longrightarrow$	**X** (0/1,-)	**Y** (0/2,-)
2:	**X** (0/1,-)	$\longrightarrow$	a (0/11,-)	
3:	**Y** (0/2,-)	$\longrightarrow$	c (0/21,-)	
4:	**X** (1/,-)	$\longrightarrow$	**X** (1/1,NA)	**Z** (1/2,-)
5:	**Z** (1/2,-)	$\longrightarrow$	b (1/21,-)	

Bedingungen an die TAG beim Algorithmus von K. Harbusch

- Die Grammtik ist in Normalform, d.h. der kontextfreie Kern ist in Chomsky-Normalform.

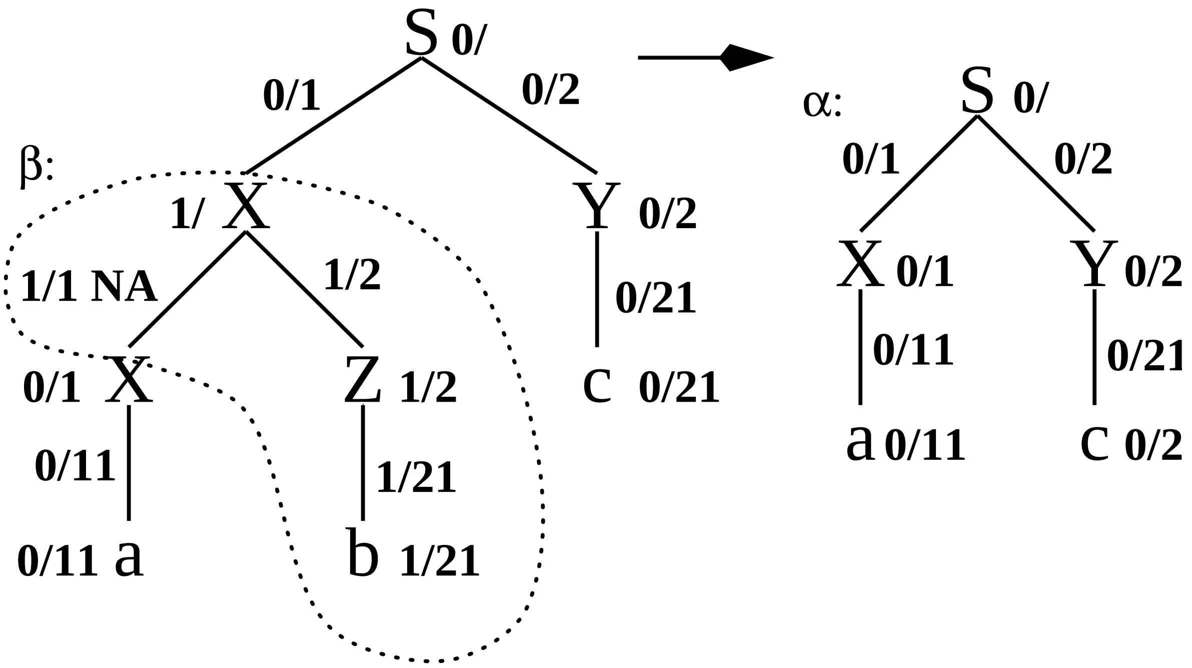
- Jeder Fußknoten eines aux. Baumes trägt ein NA-Constraint.

Erweiterung des CYK-Algorithmus



Die Annotationen der linken Seite der angewendeten Regel stehen an den Knoten, die der rechten Seite an den ausgehenden Kanten.

Entfernung eines vollständigen Baumes



Mit Hilfe der zusätzlichen Beschriftungen kann der vollständige Baum β erkannt und eliminiert werden.
Übrig bleibt der vollständige Baum α.

Innerste Bäume i-ter Stufe

Der Algorithmus von K. Harbusch

Die Eingabe w habe die Länge n:

(tag1) Behandlung des leeren Wortes

(tag2) Vergabe von Knotennummern

(tag3) Ber. des KFK mit Annotationen

(tag4) erweiterte CYK-Analyse mit w

**(tag5) Iterative Ber. der innersten Bäume der Stufe i $(1 \leq i \leq n)$
Abbruch, falls Initialbaum gefunden**

(tag6) Ausgabe des Ergebnisses