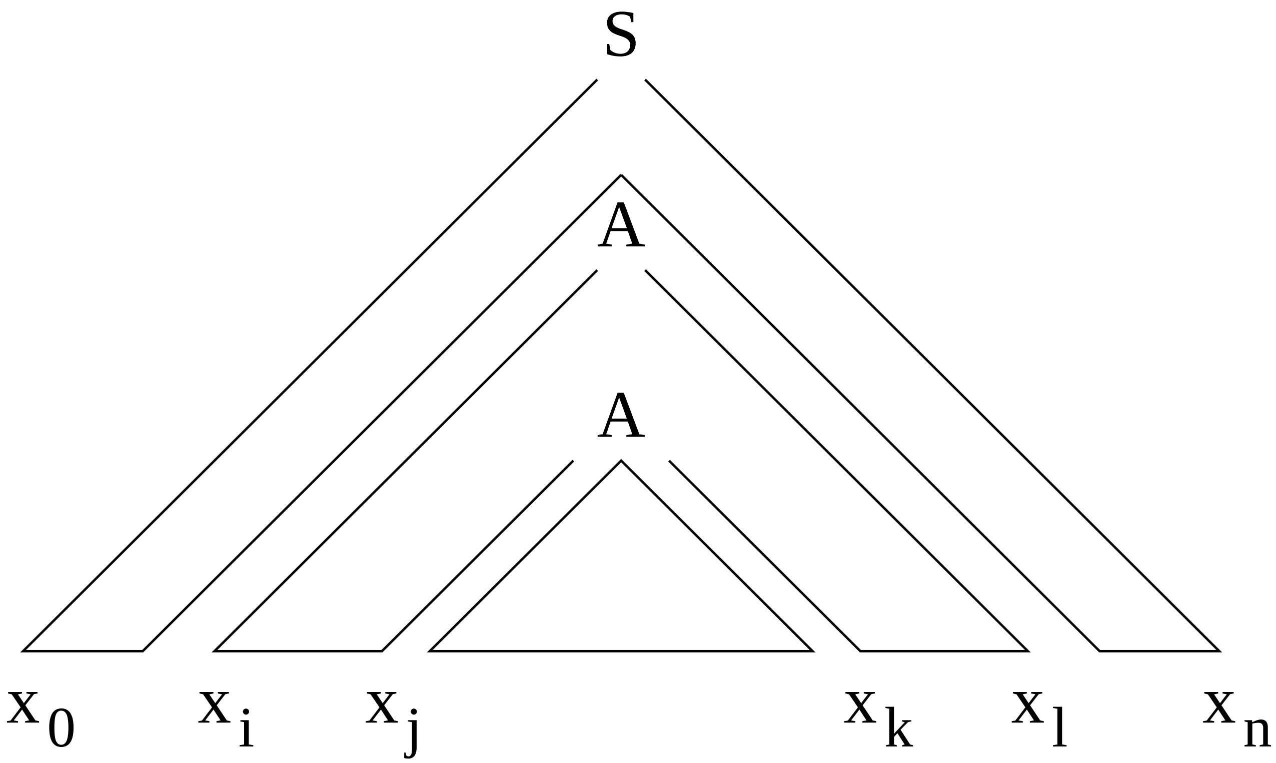
CYK Parsing für TAGs

- Erster Parsing-Algorithmus für TAGs
- Vijay-Shanker 1987
- weitere Parsing-Algorithmen
  auf Basis von Earley:
  Schabes 1990, Nederhof 1997

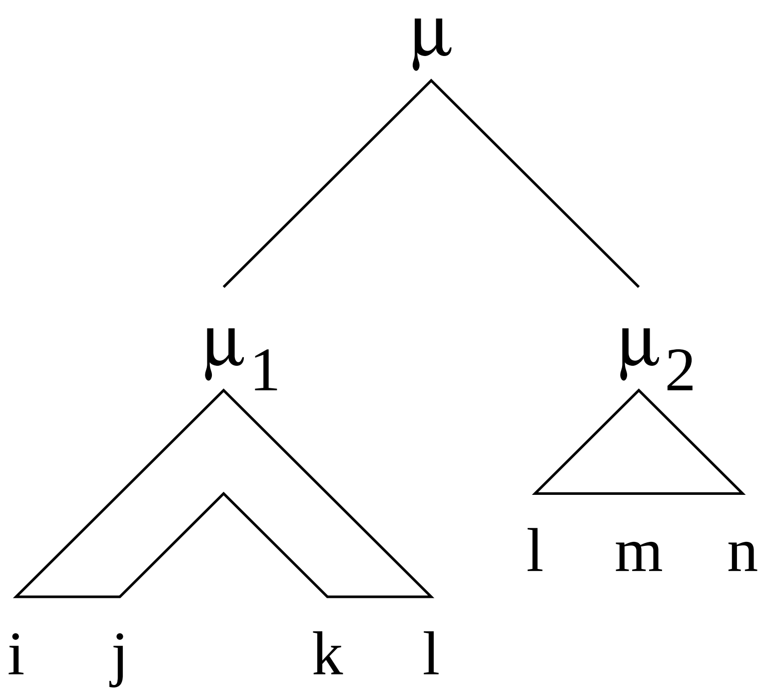
CYK für TAGs

- Annahme: TAG in Normalform
- jeder Knoten hat höchstens zwei Nachfolger
- jeder (elementare) Baum hat mindestens ein Terminal
- 4-dimensionale Matrix $M[i, j, k, l]$



CYK für TAGs

- Case 1: μ_1 ist Vorgänger des Fußknoten, μ_2 die rechte Schwester, μ der gemeinsame Mutterknoten. Wenn $\mu_1 \in M[i,j,k,l]$ und $\mu_2 \in M[l,m,m,n]$ dann trage μ in $M[i,j,k,n]$ ein:



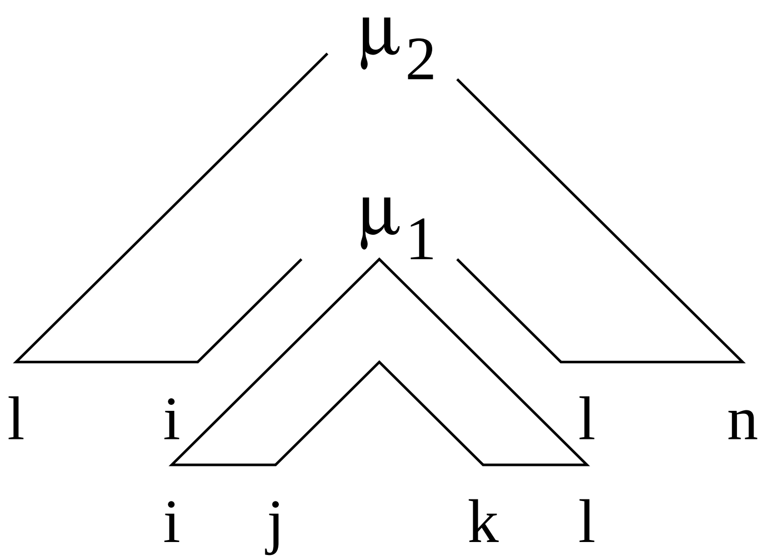
- Case 2: μ_2 ist Vorgänger des Fußknoten

- Case 3: Weder μ_1 noch μ_2 sind Vorgänger des Fußknoten

CYK für TAGs

- Case 4: Wenn Knoten $\mu_1 \in M[i,j,k,l]$ keine Schwestern hat, dann trage Vorgänger μ auch in $M[i,j,k,l]$ ein.

- Case 5: Wenn Knoten $\mu_1 \in M[i,j,k,l]$ und Wurzelknoten $\mu_2 \in M[m,i,l,n]$ gleiches Label haben, dann trage μ_1 in $M[m,j,k,n]$ ein:



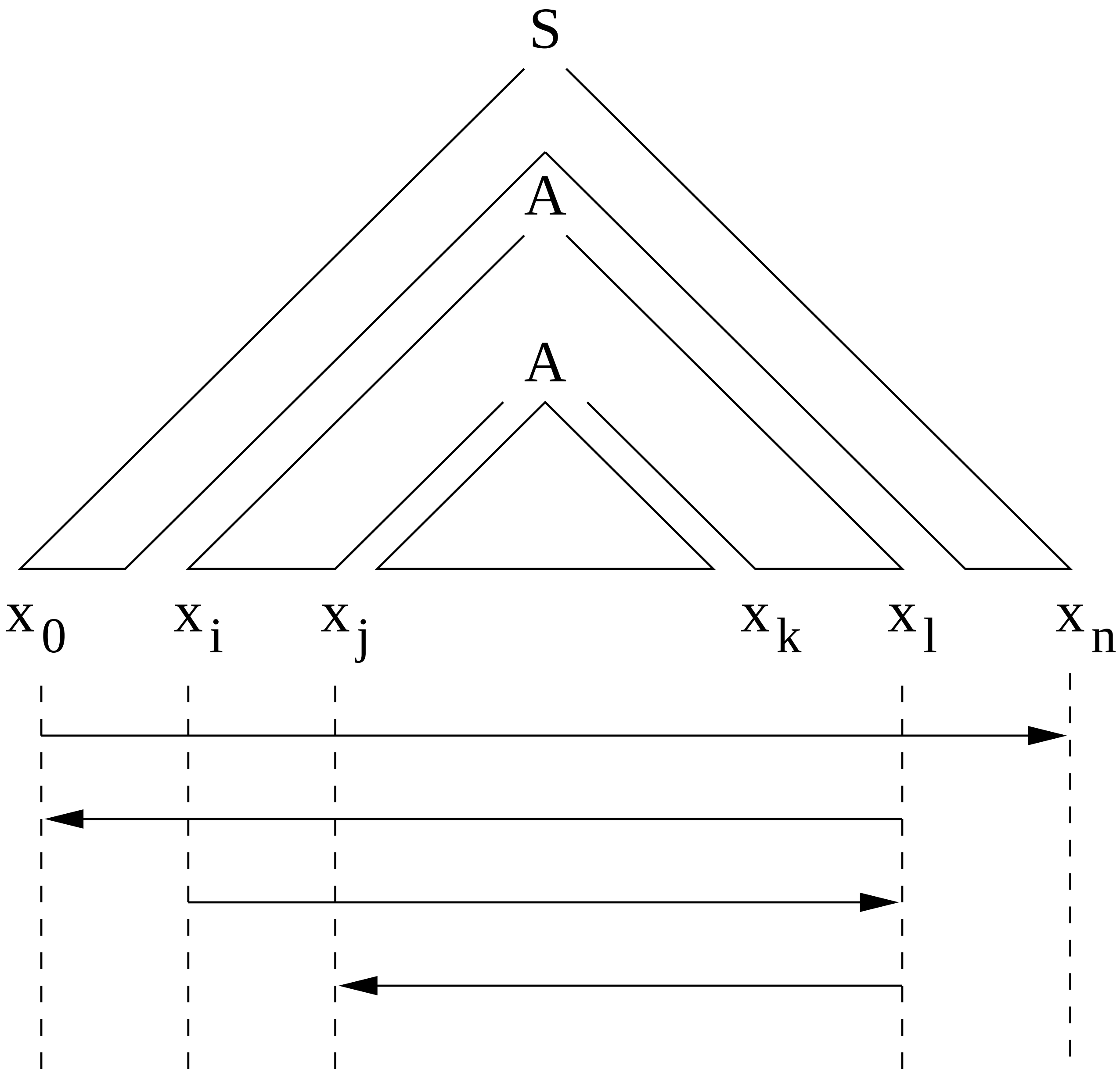
CYK für TAGs

- Input = $x_1 ... x_n$

-
```
for i := 0 to n-1 do
  put A in M[i, i+1, i+1, i+1] when A → x1
for i := 0 to n-1 do
  for j := i to n-1 do
    put all footnodes in M[i, i, j, j]
for l := 0 to n do
  for i := l downto 0 do
    for j := i to l do
      for k := l downto j do
        execute cases 1,2,3,5,4
accept if root of initial tree ∈ M[0, j, j, n],
0 ≤ j ≤ n
```

CYK für TAGs

Abarbeiten der Matrix:



CYK für TAGs

- Speicherkomplexität: $O(n^4)$
- Zeitkomplexität: $O(n^6)$
- Nachteile:
  best-case = worst-case
  braucht Normalform
- Vorteil: sehr einfach

CYK für TAGs

- Leicht erweiterbar für:
  - Substitution
  - Constraints (NA,OA,SA)
  - Merkmalsstrukturen