Semiring Algorithmus

Martina Trognitz

Xiaoxi Pang

Index

- CKY und Parsewald
- Probabilistischer Parsing-Algorithmus
  - -- Count-Algorithmus
  - -- Inside-Algorithmus
  - -- Viterbi-Algorithmus
- drei weitere probabilistische Parsingverfahren

  Viterbi-Parse-/ N-Best- / N-Best-Parse- Algorithmus
- Semiring-Parsing-Algorithmus

CKY & Parsewald

- Symbolischer Parsing-Algorithmus
- CKY: Information über Parsewald (leer ?)
- Parsewald: Aufbau der Syntaxbäume

Eingabe (CKY & Parsewald)

- Grammatik CNF(Chomsky Normal Form)
- Satz $w = w_1 \cdots w_n (n \geqslant 1)$

CKY-Algorithmus

chart[s, A, s+l] $\in$ { TRUE, FALSE }

und zwar:

$$\text{chart}[s, A, s+l] = \begin{cases} \textit{TRUE} \text{ falls } A \Rightarrow^* w_s \ldots w_{s+l-1} \text{ nicht leer} \\ \textit{False} \quad \textit{sonst} \end{cases}$$

CKY-Algorithmus

```
1.   chart[., ., .]=0;
2.   for each s:=1,…,n do
3.       for each A→w_s do
4.           chart[s, A, s+1] := TRUE;
5.   for each l := 2,…,n do
6.       for each s := 1,…,n+1-l do
7.           for each t := 1,…,l-1 do
8.               for each A → B C do
9.                   chart[s, A, s+l] := chart[s, A, s+l] ∨
10.                      chart[s, B, s+t] ∧ chart[s+t, C, s+l]
```

CKY-Algorithmus

```
1.   chart[., ., .]=0;
2.   for each s:=1,…,n do                          } 1st. Loop
3.       for each A→w_s do                        }
4.           chart[s, A, s+1] := TRUE;            }
5.   for each l := 2,…,n do
6.       for each s := 1,…,n+1-l do
7.           for each t := 1,…,l-1 do
8.               for each A → B C do
9.                   chart[s, A, s+l] := chart[s, A, s+l] ∨
10.                      chart[s, B, s+t] ∧ chart[s+t, C, s+l]
```

CKY-Algorithmus

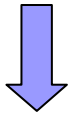
```
1.   chart[., ., .]=0;
2.   for each s:=1,…,n do                                  } 1st. Loop
3.       for each A→w_s do                                 }
4.           chart[s, A, s+1] := TRUE;                     }
5.   for each l := 2,…,n do                                } 2nd. Loop
6.       for each s := 1,…,n+1-l do                        }
7.           for each t := 1,…,l-1 do                      }
8.               for each A → B C do                       }
9.                   chart[s, A, s+l] := chart[s, A, s+l] ∨
10.                      chart[s, B, s+t] ∧ chart[s+t, C, s+l]
```

Ausgabe (CKY)

- Die Parsechart *chart* [s, A, s+l] $\in$ {TRUE, FALSE} für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Grammatikkategorien A

Parsewald-Algorithmus

- Syntaxbäume aufzubauen



$$\textit{forest}[s,A,s+l] := A \Rightarrow^{*} w_{s} \ldots w_{s+l-1}$$

Parsewald-Algorithmus

1. forest[., ., .]=∅;
2. **for each** s:=1,…,n **do**
3. **for each** A→w_s **do**
4. forest[s, A, s+1] := {A→ w_s};
5. **for each** l := 2,…,n **do**
6. **for each** s := 1,…,n+1-l **do**
7. **for each** t := 1,…,l-1 **do**
8. **for each** A → B C **do**
9. forest[s, A, s+l] := forest[s, A, s+l]+
10. A

forest[s, B, s+t] forest[s+t, C, s+l]

Parsewald-Algorithmus

```
1.    forest[., ., .]=∅;
2.    for each s:=1,…,n do
3.        for each A→w_s do
4.            forest[s, A, s+1] := {A→ w_s};
5.    for each l := 2,…,n do
6.        for each s := 1,…,n+1-l do
7.            for each t := 1,…,l-1 do
8.                for each A → B C do
9.                    forest[s, A, s+l] := forest[s, A, s+l]+
10.                                          A
                                        forest[s, B, s+t]   forest[s+t, C, s+l]
```

Parsewald-Algorithmus

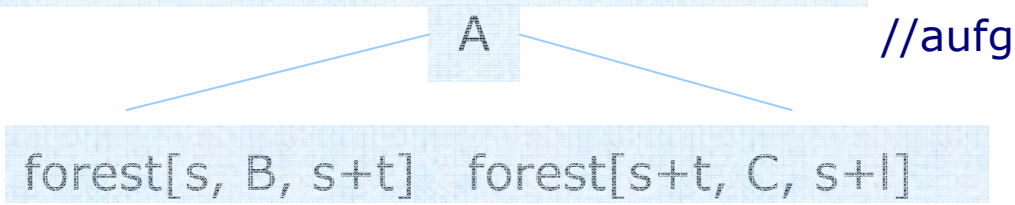
```
1.    forest[., ., .]=∅;                    // forest[ ] als leere Menge initialisiert
2.    for each s:=1,…,n do
3.        for each A→w_s do
4.            forest[s, A, s+1] := {A→ w_s};
5.    for each l := 2,…,n do
6.        for each s := 1,…,n+1-l do
7.            for each t := 1,…,l-1 do
8.                for each A → B C do
9.                    forest[s, A, s+l] := forest[s, A, s+l]+
10.                                          A
                                    forest[s, B, s+t]   forest[s+t, C, s+l]
```

Parsewald-Algorithmus

```
1.    forest[., ., .]=∅;                    // forest[ ] als leere Menge initialisiert
2.    for each s:=1,…,n do
3.        for each A→w_s do
4.            forest[s, A, s+1] := {A→ w_s}; //Terminalsymbols zugeordnet
5.    for each l := 2,…,n do
6.        for each s := 1,…,n+1-l do
7.            for each t := 1,…,l-1 do
8.                for each A → B C do
9.                    forest[s, A, s+l] := forest[s, A, s+l]+
10.                                          A
                                    forest[s, B, s+t]   forest[s+t, C, s+l]
```

Parsewald-Algorithmus

1. forest[., ., .]=∅; // forest[ ] als leere Menge initialisiert
2. **for each** s:=1,…,n **do**
3. **for each** A→w_s **do**
4. forest[s, A, s+1] := {A→ w_s}; //Terminalsymbols zugeordnet
5. **for each** l := 2,…,n **do**
6. **for each** s := 1,…,n+1-l **do**
7. **for each** t := 1,…,l-1 **do**
8. **for each** A → B C **do**
9. forest[s, A, s+l] := forest[s, A, s+l]+ //Syntaxbäume
10. //aufgebaut



Ausgabe (Parsewald)

- Die Parsewälder *forest* [s, A, s+l] für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Wurzelkategorien A

Laufzeit

- Die Laufzeit für beide Algorithmus ist $O(n^3)$

  1st. Loop $\Rightarrow O(n)$

  2nd. Loop $\Rightarrow O(n^3)$

  $\Rightarrow O(n^3)$

Count-Algorithmus

Count-Algorithmus

Eingabe

- Symbolische Grammatik in CNF
- Satz $w = w_1 \cdots w_n$ $(n \geqslant 1)$

Was kann Count-Algorithmus tun?

- Verwalten die Häufigkeiten, die Zahl der Analysen aller Teilsätze

  ---- deswegen gehört Count-Altorithmus nicht mehr zu dem symbolischen Parsing aber probabilistischen Parsing

Count-Algorithmus

Ausgabe

- count [s, A, s+l] := | $A \Rightarrow^* w_s \ldots w_{s+l-1}$ |

  das bedeutet:

  Die Anzahl *count* [s, A, s+l] der grammatischen Analysen für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Grammatikkategorien A

Count-Algorithmus

```
1.   count[., ., .]=0;
2.   for each s:=1,…,n do
3.       for each A→w_s do
4.           count[s, A, s+1] := 1;
5.   for each l := 2,…,n do
6.       for each s := 1,…,n+1-l do
7.           for each t := 1,…,l-1 do
8.               for each A → B C do
9.                   count[s, A, s+l] := count[s, A, s+l]+
10.                      count[s, B, s+t] · count[s+t, C,s+l]
```

Count-Algorithmus

```
1.    count[., ., .]=0;
2.    for each s:=1,…,n do
3.        for each A→w_s do
4.            count[s, A, s+1] := 1;
5.    for each l := 2,…,n do
6.        for each s := 1,…,n+1-l do
7.            for each t := 1,…,l-1 do
8.                for each A → B C do
9.                    count[s, A, s+l] := count[s, A, s+l]+
10.                       count[s, B, s+t] · count[s+t, C,s+l]
```

Count-Algorithmus

```
1.   count[., ., .]=0;          //count[ ] mit NULL initialisiert
2.   for each s:=1,…,n do
3.       for each A→w_s do
4.           count[s, A, s+1] := 1;
5.   for each l := 2,…,n do
6.       for each s := 1,…,n+1-l do
7.           for each t := 1,…,l-1 do
8.               for each A → B C do
9.                   count[s, A, s+l] := count[s, A, s+l]+
10.                    count[s, B, s+t] · count[s+t, C,s+l]
```

Count-Algorithmus

```
1.   count[., ., .]=0;                    //count[ ] mit NULL initialisiert
2.   for each s:=1,…,n do                 // 1st. Loop:  Terminalsymbols
3.       for each A→w_s do                zugeordenet, dann count[] vom
4.           count[s, A, s+1] := 1;       Wert 1 zugewiesen
5.   for each l := 2,…,n do
6.       for each s := 1,…,n+1-l do
7.           for each t := 1,…,l-1 do
8.               for each A → B C do
9.                   count[s, A, s+l] := count[s, A, s+l]+
10.                    count[s, B, s+t] · count[s+t, C,s+l]
```

Count-Algorithmus

```
1.   count[., ., .]=0;                    //count[ ] mit NULL initialisiert
2.   for each s:=1,…,n do
3.       for each A→w_s do
4.           count[s, A, s+1] := 1;
5.   for each l := 2,…,n do
6.       for each l := 2,…,n do
7.           for each l := 2,…,n do
8.               for each A → B C do
9.                   count[s, A, s+l] := count[s, A, s+l]+
10.                    count[s, B, s+t] · count[s+t, C,s+l]
```

// 1st. Loop: Terminalsymbols zugeordenet, dann count[] vom Wert 1 zugewiesen

2nd. Loop:

A → B C

Wenn eine Regel gefunden, eins ins count[] gezählt

Laufzeit

- gleiche Loop Anweisungen mit die in CKY- und Parsewald-Algorithmus, somit ist die Laufzeit bei Count-Algorithmus auch $O(n^3)$

Warum Multiplikation?

- In Zeile 10 wird die Multiplikation von count der zwei Teilbäume gerechnet :

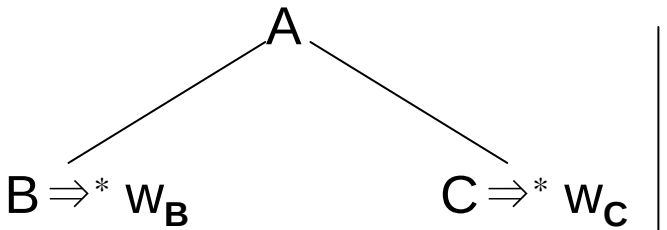
  count[s, A, s+l] := count[s, A, s+l]+

  count[s, B, s+t] · count[s+t, C,s+l]

Warum Multiplikation?

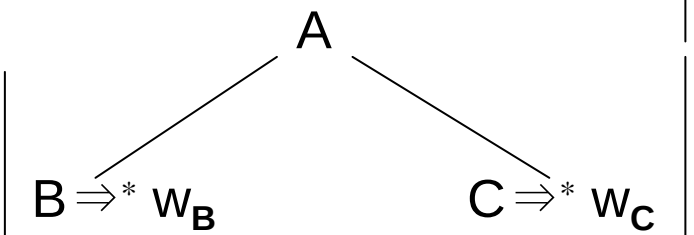
$\text{count}(A \Rightarrow^* w_A) := |\, A \Rightarrow^* w_A \,|$

$$= \left| \sum_{A \to BC,\ w_A = w_B w_C} \right.$$



$$\left. \right|$$

$$= \sum_{A \to BC,\ w_A = w_B w_C}$$



$$= \sum_{A \to BC,\ w_A = w_B w_C} |B \Rightarrow^* w_B| \cdot |C \Rightarrow^* w_C|$$

$$= \sum_{A \to BC,\ w_A = w_B w_C} \text{count}(B \Rightarrow^* w_B) \cdot \text{count}(C \Rightarrow^* w_C)$$

Warum Multiplikation?

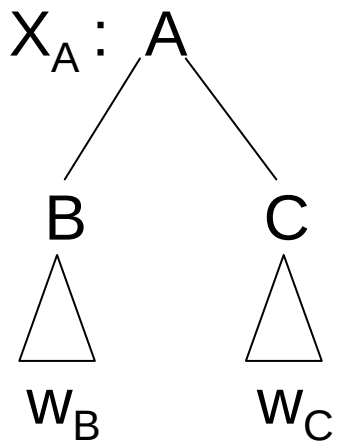
count[s, A, s+l] := count[s, A, s+l]+

count[s, B, s+t] · count[s+t, C,s+l]

Kreuzungsprodukte

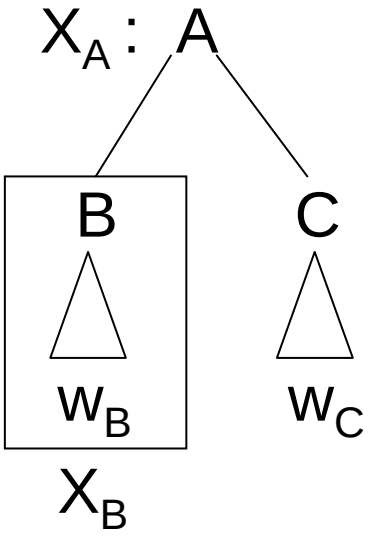
Warum Multiplikation?

z.B Baum X_A $A \Rightarrow^* w_A ; A \rightarrow B\ C ; B \Rightarrow^* w_B ; C \Rightarrow^* w_C ; w_A \Rightarrow w_B\ w_C$



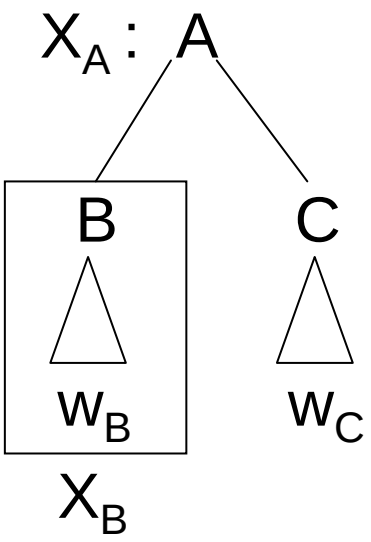
Warum Multiplikation?

z.B Baum X_A $A \Rightarrow^* w_A ; A \rightarrow B\ C ; B \Rightarrow^* w_B ; C \Rightarrow^* w_C ; w_A \Rightarrow w_B\ w_C$

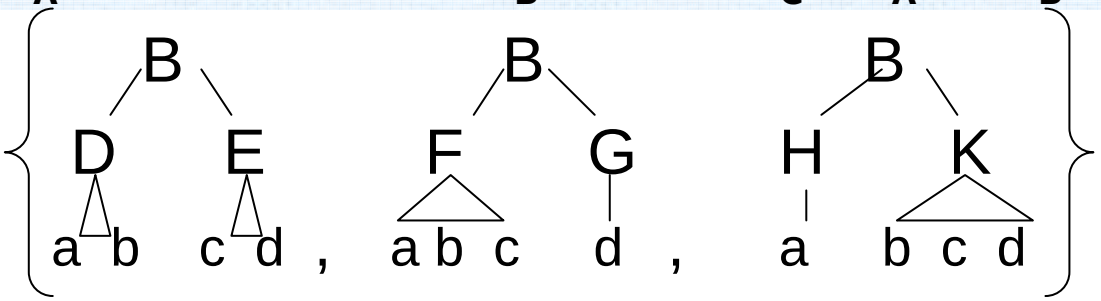


Warum Multiplikation?

z.B Baum X_A $A \Rightarrow^* w_A ; A \rightarrow B\ C ; B \Rightarrow^* w_B ; C \Rightarrow^* w_C ; w_A \Rightarrow w_B\ w_C$

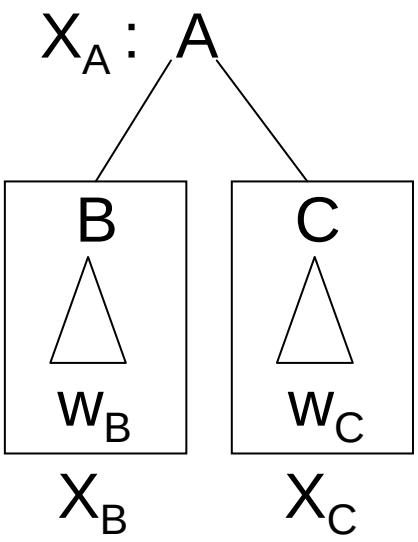


$X_B =$



Warum Multiplikation?

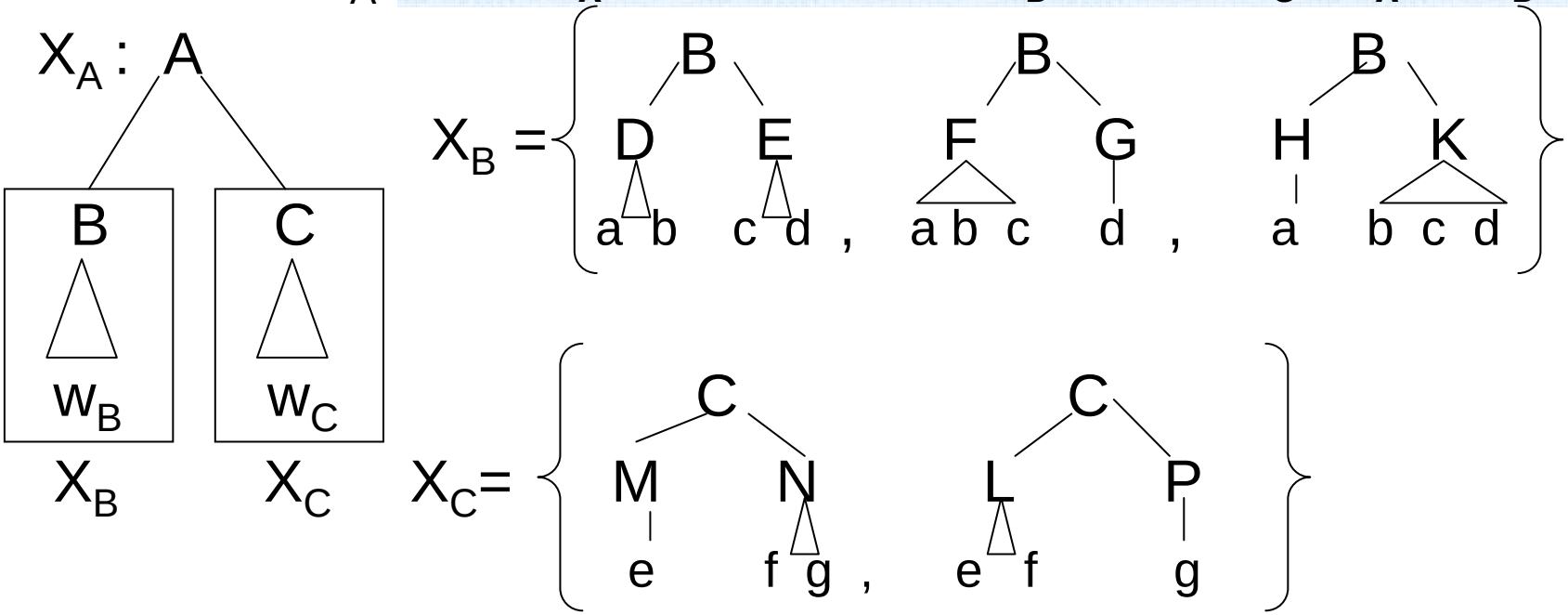
z.B Baum X_A $A \Rightarrow^* w_A ; A \rightarrow B\ C ; B \Rightarrow^* w_B ; C \Rightarrow^* w_C ; w_A \Rightarrow w_B\ w_C$



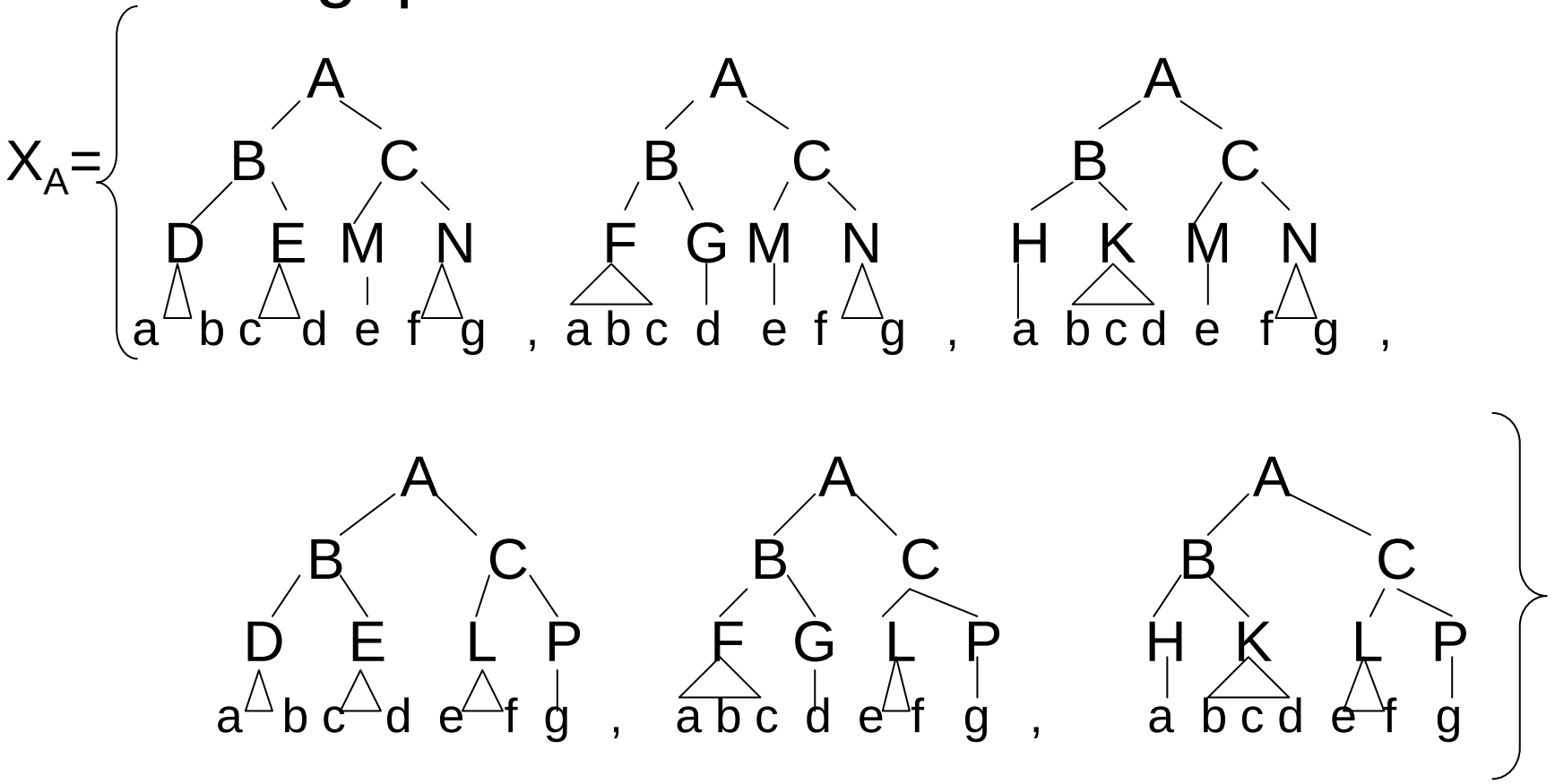
$X_B =$ { B(D(a b), E(c d)) , B(F(a b c), G(d)) , B(H(a), K(b c d)) }

Warum Multiplikation?

z.B Baum X_A $A \Rightarrow^* w_A ; A \rightarrow B\,C ; B \Rightarrow^* w_B ; C \Rightarrow^* w_C ; w_A \Rightarrow w_B\, w_C$



Kreuzungsprodukte:

$X_A =$ 

$\Rightarrow$ count($A \Rightarrow^* w_A$) = count($B \Rightarrow^* w_B$) · count($C \Rightarrow^* w_C$) = 3*2 =6

They study fish in cans.

Grammatik:

Lexikon

intr → {sleep, fish}

trans →{study, visit}

N →{they, cans, fish}

prep →{in, by, with}

Regeln

S →N intr

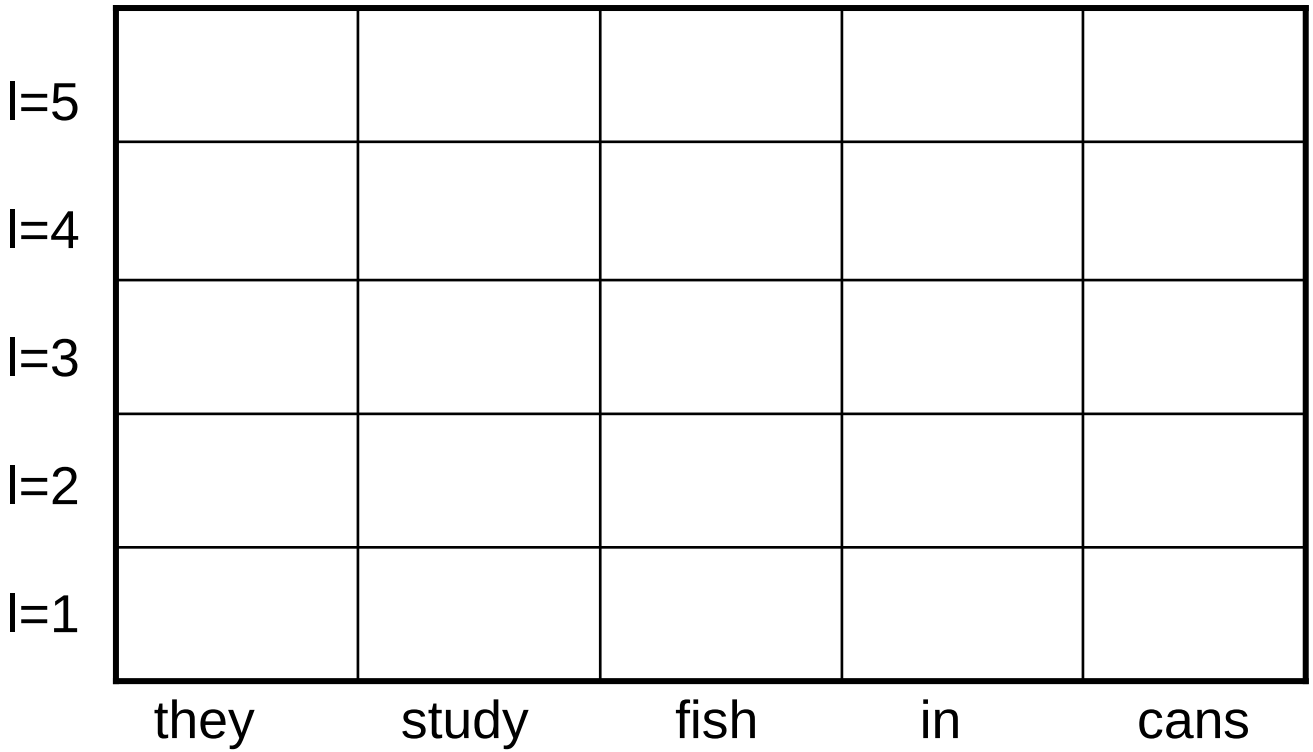
intr →trans N

intr →intr PP

N →N PP

PP →prep N

```
count [.,.,. ]=0;
```



Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

s=1

for each s:=1,…,n **do**
  for each A→w_s **do**

l=5					
l=4					
l=3					
l=2					
l=1	[1 ,N, 2]=1				
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

s=2

for each s:=1,…,n **do**
  for each A→w_s **do**

l=5					
l=4					
l=3					
l=2					
l=1	[1 ,N, 2]=1	[2,trans,3]=1			
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

s=3

for each s:=1,…,n **do**
  for each A→w_s **do**

l=5					
l=4					
l=3					
l=2					
l=1	**[1, N, 2]=1**	**[2,trans,3]=1**	**[3,intr,4]=1** **[3, N, 4]=1**		
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

s=4

for each s:=1,…,n **do**
  for each A→w_s **do**

l=5					
l=4					
l=3					
l=2					
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

s=5

for each s:=1,…,n **do**
  for each A→w_s **do**

l=5					
l=4					
l=3					
l=2					
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

Ende der ersten Schleife…

l=5					
l=4					
l=3					
l=2					
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=2;

s=1;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2					
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=2;

s=1;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2					
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=2;

s=2;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2					
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=2;

s=2;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2		[2,intr,4]=1			
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=2;

s=3 bis 4

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=3;

s=1;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=3;

s=1;

t=2;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=3;

s=1;

t=2;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3					
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=3;

s=2 bis 3; (auf gleiche Weise läuft es bis zum Ende dieser Zeile)

t=1bis 2;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=4;

s=1;

t=1 bis 3;(nur leere Felder treffen)

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=4;

s=1;

t=1 bis 3;(nur leere Felder treffen)

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=4;

s=2;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4					
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=4;

s=2;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4		[2,intr,6]=1			
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=4;

s=2;

t=2;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5					
l=4		[2,intr,6]=1 [2,intr,6]=1			
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=5;

s=1;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5	[1, S, 6]=1				
l=4		[2,intr,6]=1 [2,intr,6]=1			
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=5;

s=1;

t=1;

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5	[1, S, 6]=1+1				
l=4		[2,intr,6]=1 [2,intr,6]=1			
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

l=5;

s=1;

t=2 bis 4; (nur leere Felder treffen)

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5	[1, S, 6]=2				
l=4		[2,intr,6]=1 [2,intr,6]=1			
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

Ende der zweiten Schleife

```
for each l := 2,…,n do
    for each s := 1,…,n+1-l do
        for each t := 1,…,l-1 do
            for each A → B C do
```

l=5	[1, S, 6]=2				
l=4		[2,intr,6]=1 [2,intr,6]=1			
l=3			[3,intr,6]=1 [3, N, 6]=1		
l=2		[2,intr,4]=1		[4, PP, 6]=1	
l=1	[1, N, 2]=1	[2,trans,3]=1	[3,intr,4]=1 [3, N, 4]=1	[4,trans,5]=1	[5, N, 6]=1
	they	study	fish	in	cans

Lexikon:
intr → {sleep, fish}
trans →{study, visit}
N →{they, cans, fish}
prep →{in, by, with}
Regeln:
S → N intr
intr → trans N
intr → intr PP
N → N PP
PP → prep N

Ausgabe: count [1, S, 6] = 2