plan

- answer questions
- left-corner-parser
- review last slide (inside-outside algorithm)

Fragen

1) Folie 102&103: Kannst du bitte nochmal das Beispiel zu Kneser Ney erklären?
Ich verstehe nicht woher f*(a) = |{(a,a)}| = 1 kommt im Vergleich zum Standard-Backoff-Verfahren

2) Kannst du die ersten 2 Formeln auf Folie 164 zum Forward-Backward-Algorithmus nochmal erklären?

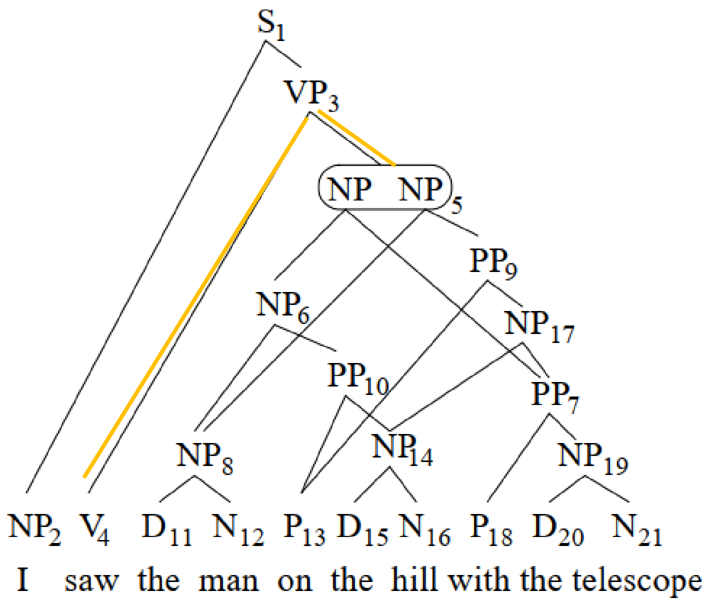
3) Folie 211: Bsp Outside-Algorithmus. Wie genau geht man mit den beiden NPs (NP5) um?

4) Folie 206: Viterbi-Beispiel -> ist delta(the) = 3 weil "the" 3mal in dem Satz vorkommt, oder wird jedes Terminalsymbol automatisch mit 1 initialisiert?

jedes Terminalsymbol automatisch mit 1 initialisiert

Folie 244: Berkeley Parser -> Wann setzt man bei den synthetischen Merkmalen 0 oder 1?

3) Folie 211: Bsp Outside-Algorithmus. Wie genau geht man mit den beiden NPs (NP5) um?



outside(NP5) = out(VP3 -> V4, NP5) # (1)

= out(VP3) p(VP -> V, NP) in(V4) #(2)

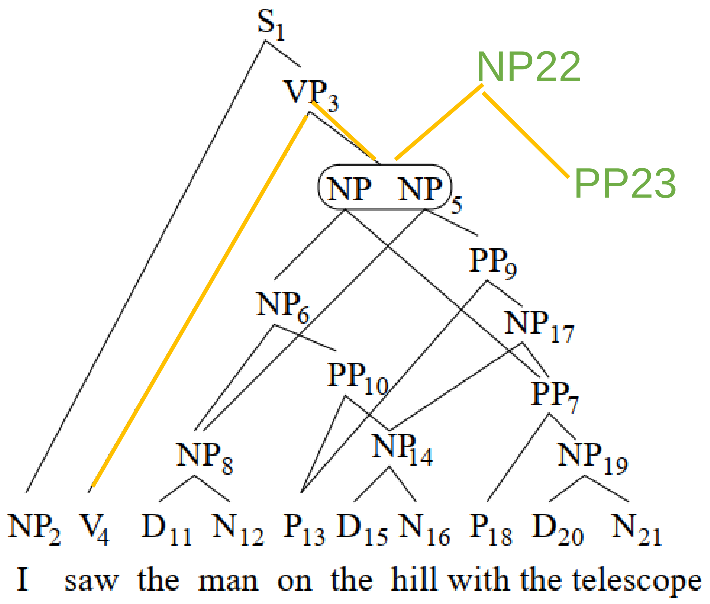
Outside-Algorithmus

$$\beta(S) = 1 \quad \text{für Startsymbol S}$$

(1) $$\beta(A) = \sum_{B \to \gamma A \delta} \beta(B \to \gamma \underline{A} \delta)$$

(2) $$\beta(B \to X_1 ... X_m \underline{A} X_{m+1} ... X_n) = \beta(B) p(B \to X_1 ... X_m A X_{m+1} ... X_n) \prod_{i=1}^{n} \alpha(X_i)$$

3) Folie 211: Bsp Outside-Algorithmus. Wie genau geht man mit den beiden NPs (NP5) um?



In case NP5 has 2 parent nodes, we compute the following

outside(NP5) = out(VP3 -> V4, NP5) + out(NP22 -> NP5, PP23)
= out(VP3) p(VP -> V, NP)in(V4) + out(NP22) p(NP -> NP, PP) in(PP23)

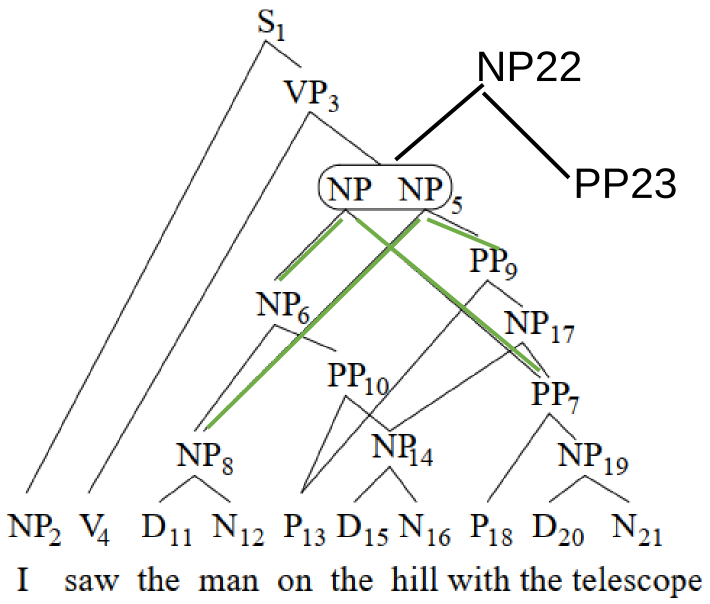
Outside-Algorithmus

$$\beta(S) = 1 \quad \text{für Startsymbol S}$$

$$\beta(A) = \sum_{B \to \gamma A \delta} \beta(B \to \gamma \underline{A} \delta)$$

$$\beta(B \to X_1 ... X_m \underline{A} X_{m+1} ... X_n) = \beta(B) p(B \to X_1 ... X_m A X_{m+1} ... X_n) \prod_{i=1}^{n} \alpha(X_i)$$

3) Folie 211: Bsp Outside-Algorithmus. Wie genau geht man mit den beiden NPs (NP5) um?



compare to Inside-WK

inside(NP5) = in(NP5 -> NP6, PP7) + in(NP5 -> NP8, PP9) #(2)
= p(NP -> NP, PP) in(NP6) in(PP7) + p(NP ->NP, PP) in(NP8) in(PP9) #(1)

Inside-Wahrscheinlichkeiten:

$$\alpha(a) = 1 \quad \text{für Terminalsymbol a}$$

(1) $$\alpha(A \rightarrow X_1 ... X_n) = p(A \rightarrow X_1 ... X_n) \prod_{i=1}^{n} \alpha(X_i) \text{ für Parsewaldregeln}$$

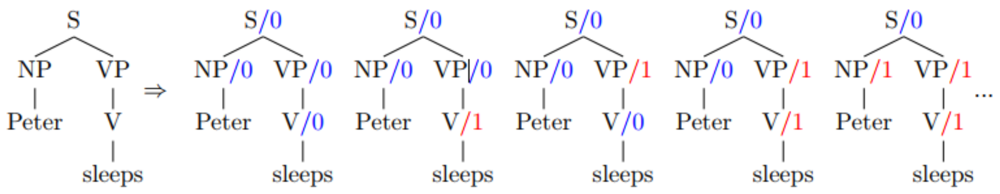
(2) $$\alpha(A) = \sum_{A \rightarrow \gamma} \alpha(A \rightarrow \gamma) \quad \text{für Nichtterminale A}$$

Folie 244: Berkeley Parser -> Wann setzt man bei den synthetischen Merkmalen 0 oder 1?

Synthetische Merkmale

Grundidee (von Petrov/Klein)

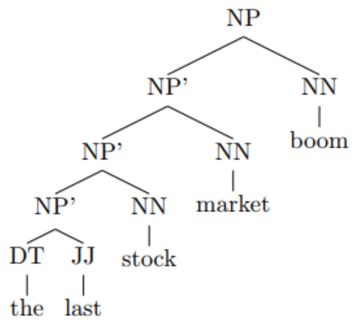
- Alle Kategorien werden durch ein synthetisches Merkmal mit den Werten 0 bzw. 1 aufgespalten.
- Jeder Parse der Baumbank kann von der neuen Grammatik auf viele unterschiedliche Arten generiert werden.
- Durch EM-Training wird die neue Grammatik an die Baumbank angepasst.



Die modifizierte Grammatik liefert für den alten Parsebaum $2^4 = 16$ neue Parsebäume.

See an example on the next page

Vorverarbeitung



1. Binarisierung der Parsebäume
2. Extraktion einer Grammatik mit Häufigkeiten

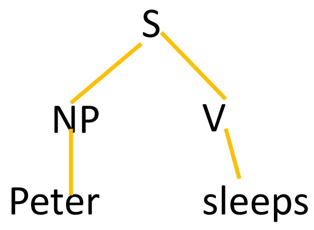
NP	$\rightarrow$ NP' NN	1
NP'	$\rightarrow$ NP' NN	2
NP'	$\rightarrow$ DT JJ	1

3. Aufspaltung jeder Kategorie in 2 neue Kategorien
   Uniforme Verteilung der Häufigkeiten über die neuen Regeln
   (mit kleinen Abweichungen, um die Symmetrie zu brechen.
   Sonst bleibt das EM-Training im Anfangszustand stecken.)

...

NP'/0	$\rightarrow$ NP'/0 NN/0	0.24
NP'/0	$\rightarrow$ NP'/0 NN/1	0.26
NP'/0	$\rightarrow$ NP'/1 NN/0	0.25

...



Original grammar

S -> NP V

NP -> Peter

V -> sleeps

----

split V into 2 categories, we get

V/1 -> sleeps

V/0 -> sleeps

split NP into 2 categories, we get

NP/1 -> Peter

NP/0 -> Peter

split S into 2 categories, we get

S1/1 -> NP/1 V/1

S1/1 -> NP/1 V/0

S1/1 -> NP/0 V/1

S1/1 -> NP/0 V/0

S1/0 -> NP/1 V/1

S1/0 -> NP/1 V/0

S1/0 -> NP/0 V/1

S1/0 -> NP/0 V/0

1) Folie 102&103: Kannst du bitte nochmal das Beispiel zu Kneser Ney erklären?
Ich verstehe nicht woher f*(a) = |{(a,a)}| = 1 kommt im Vergleich zum Standard-Backoff-Verfahren

Beispiel

Gegebene Häufigkeiten

f(a,a) = 1   f(b,a) = 0
f(a,b) = 2   f(b,b) = 1

Discount-Berechnung

$N_1 = 2$ $\delta = 2\ /(2 + 2{*}1) = 0.5$
$N_2 = 1$

Berechnung der Backoff-Wahrscheinlichkeitsverteilung

Standard-Backoff-Verfahren
f(a) = f(a,a) + f(b,a) = 1
f(b) = f(a,b) + f(b,b) = 3
p(a) = f(a) / (f(a) + f(b)) = 1/4
p(b) = f(b) / (f(a) + f(b)) = 3/4

Kneser-Ney-Verfahren
$f^*(\text{a}) = |\{(a,a)\}| = 1$
$f^*(\text{b}) = |\{(a,b),(b,b)\}| = 2$
$\text{p(a)} = f^*(\text{a}) \ / \ (f^*(\text{a}) + f^*(\text{b})) = 1/3$
$\text{p(b)} = f^*(\text{b}) \ / \ (f^*(\text{a}) + f^*(\text{b})) = 2/3$

Berechnung der relativen Häufigkeiten mit Discount (Standard)

r(a|a) = max(0,f(a,a)-δ)/(f(a,a)+f(a,b)) = max(0, (1-0.5))/(1+2) = 1/6
r(b|a) = max(0,f(a,b)-δ)/(f(a,a)+f(a,b)) = max(0, (2-0.5))/(1+2) = 3/6

r(a|b) = max(0,f(b,a)-δ)/(f(b,a)+f(b,b)) = max(0, (0-0.5))/(0+1) = 0
r(b|b) = max(0,f(b,b)-δ)/(f(b,a)+f(b,b)) = max(0, (1-0.5))/(0+1) = 0.5

Kneser-Ney Backoff-Verteilung

Bei der bisherigen Berechnung der n-1-Gramm-Häufigkeiten zur Schätzung der Backoff-Verteilung summieren wir die Häufigkeiten über alle möglichen Vorgängerwörter w':

$$f(C,w) = \sum_{w'} f(w',C,w)$$

C ist eine (eventuell leere) Folge von Wörtern.

Bei Kneser-Ney zählen wir, wieviele **unterschiedliche** Wörter vor dem Wort-n-Gramm aufgetreten sind:

$$f^*(C,w) = \sum_{w'} \mathbf{1}_{f(w',C,w)>0}$$

$\mathbf{1}_{test}$ ist 1, falls *test* wahr ist und sonst 0.

Die Kneser-Ney-Methode zählt n-Gramm-Types (statt -Tokens).

Aus den so ermittelten Häufigkeiten, werden dann die Parameter der Backoff-Wahrscheinlichkeits-Verteilungen geschätzt.

$$p_{backoff}(w|C) = \frac{f^*(C,w)}{\sum_{w'} f^*(C,w')}$$

2) Kannst du die ersten 2 Formeln auf Folie 164 zum Forward-Backward-Algorithmus nochmal erklären?

- was bedeutet das |w| : wk = w was unter dem Summenzeichen steht?

Forward-Backward-Algorithmus

Summierung der Aposteriori-Wahrscheinlichkeiten über alle Sätze **w** im Korpus C und über alle Wortpositionen k im Satz zu erwarteten Häufigkeiten:

$$f_{tw} = \sum_{\mathbf{w} \in C} \sum_{1 \leq k \leq |\mathbf{w}| : w_k = w} \gamma_t(k, \mathbf{w})$$

for k in {1,2,3..|w|} where w_k = w

$$f_{tt'} = \sum_{\mathbf{w} \in C} \sum_{k=1}^{n+1} \gamma_{tt'}(k, \mathbf{w})$$

Der Ausdruck $\sum_{1 \leq k \leq n : w_k = w} \gamma_t(k)$ summiert über alle Positionen $k \in \{1, 2, \ldots, n\}$ mit $w_k = w$. Man kann unter Verwendung der Indikatorfunktion auch schreiben: $\sum_{1 \leq k \leq n} \gamma_t(k) \mathbb{1}_{w_k = w}$.

$\gamma_{tt'}(k, \mathbf{w})$ ist der Wert von $\gamma_{tt'}(k)$ für den Satz **w**.

Neuschätzung der HMM-Parameter (M-Schritt)

$$p(w|t) = \frac{f_{tw}}{\sum_{w'} f_{tw'}}$$

$$p(t'|t) = \frac{f_{tt'}}{\sum_{t''} f_{tt''}}$$

f(tag, word)
f(tag, tag)

Summierung der Aposteriori-Wahrscheinlichkeiten über alle Sätze **w** im Korpus C und über alle Wortpositionen k im Satz zu erwarteten Häufigkeiten:

$$f_{tw} = \sum_{\mathbf{w}\in C} \sum_{1\le k\le|\mathbf{w}|:w_k=w} \gamma_t(k,\mathbf{w})$$

$$f_{tt'} = \sum_{\mathbf{w}\in C}\sum_{k=1}^{n+1}\gamma_{tt'}(k,\mathbf{w})$$

Der Ausdruck $\sum_{1\le k\le n: w_k=w}\gamma_t(k)$ summiert über alle Positionen $k\in\{1,2,\dots,n\}$ mit $w_k = w$. Man kann unter Verwendung der Indikatorfunktion auch schreiben: $\sum_{1\le k\le n}\gamma_t(k)\mathbb{1}_{w_k=w}$.

$\gamma_{\cdot\cdot\prime}(k,\mathbf{w})$ ist der Wert von $\gamma_{\cdot\cdot\prime}(k)$ für den Satz **w**.

for each sentence in the corpus and for each position k in the sentence where the word at that position = w, sum up gamma_t(k).

Example: we want to compute f(MD, can)

For each sentence, look for positions that have the word "can".

In sentence 1, we found position2,3,5. In sentece2, we found position 5. So,

f(MD, can) = gamma_MD(k=2, sent1) +
gamma_MD(k=3, sent1) +
gamma_MD(k=5, sent1) +
gamma_MD(k=5, sent2)

0	1	2	3	4	5	6
	I	can	can	a	can	
<s>	PRO	MD	MD	DT	MD	</s>
	PN	NN	NN		NN	
		VB	VB		VB	

gamma_MD(k=3)

0	1	2	3	4	5	6
	She	has	a	small	can	
<s>	PRO	MD	DT	ADV	MD	</s>
	PN	VB				

gamma_MD(k=5)

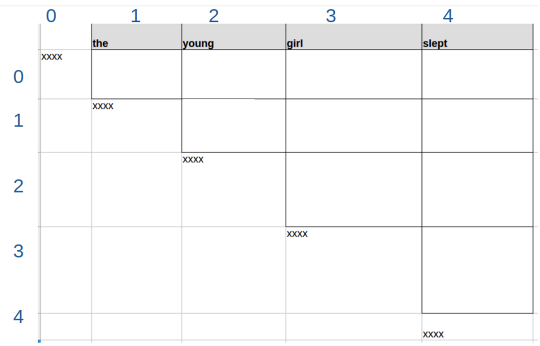
34

Left-Corner-Parser

Left-Corner-Parser: an algorithm for parsing texts (to syntactic analyses) used with CFG

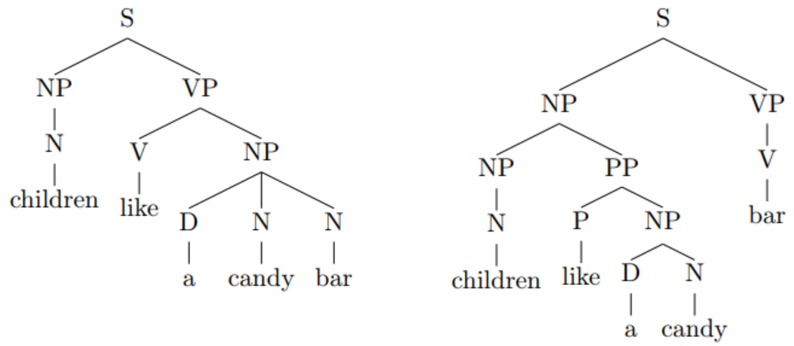
children like a candy bar

Left-Corner-Parser

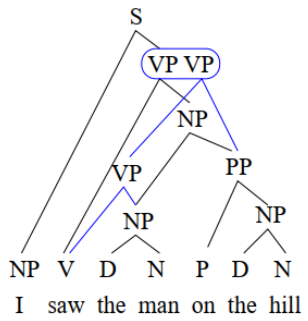


CFG

S	→	NP VP
NP	→	NP PP
NP	→	D N
NP	→	N
NP	→	D N N
NP	→	N N
VP	→	V NP
VP	→	V
PP	→	P NP
D	→	the
D	→	a
N	→	children
N	→	candy
N	→	bar
V	→	bar
V	→	like
P	→	like
P	→	for



generally, the output of such parser is a Parsewald (not 2 trees)



Left corner parser

From Wikipedia, the free encyclopedia

In computer science, a **left corner parser** is a type of chart parser used for parsing context-free grammars. It combines the top-down and bottom-up approaches of parsing. The name derives from the use of the left corner of the grammar's production rules.

An early description of a left corner parser is "A Syntax-Oriented Translator" by Peter Zilahy Ingerman.[1][2]

In computer science, a **chart parser** is a type of parser suitable for ambiguous grammars (including grammars of natural languages). It uses the dynamic programming approach—partial hypothesized results are stored in a structure called a chart and can be re-used. This eliminates backtracking and prevents a combinatorial explosion.

Chart parsing is generally credited to Martin Kay.[1]

Earley parser

From Wikipedia, the free encyclopedia

In computer science, the **Earley parser** is an algorithm for parsing strings that belong to a given context-free language, though (depending on the variant) it may suffer problems with certain nullable grammars.[1] The algorithm, named after its inventor, Jay Earley, is a chart parser that uses dynamic programming; it is mainly used for parsing in computational linguistics. It was first introduced in his dissertation[2] in 1968 (and later appeared in an abbreviated, more legible, form in a journal[3]).

Earley parsers are appealing because they can parse all context-free languages, unlike LR parsers and LL parsers, which are more typically used in compilers but which can only handle restricted classes of languages. The Earley parser executes in cubic time in the general case $O(n^3)$, where *n* is the length of the parsed string, quadratic time for unambiguous grammars $O(n^2)$,[4] and linear time for all deterministic context-free grammars. It performs particularly well when the rules are written left-recursively.

Aktionen des LC-Parsers

Scan sucht im Lexikon alle Wortarten X des nächsten Wortes w und trägt die Regel $X \rightarrow w \cdot$ ins zweit-unterste Feld ein.

Predict wird aufgerufen, wenn eine Regel $X \rightarrow \alpha\cdot$ mit dem Punkt am Ende eingetragen wurde. Predict trägt im gleichen Feld wie diese Regel alle Grammatikregeln der Form $Y \rightarrow X \cdot \beta$ ein.

Complete wird ebenfalls aufgerufen, wenn eine Regel $X \rightarrow \alpha\cdot$ mit dem Punkt am Ende eingetragen wurde. Complete durchsucht alle Punktregeln in der Chart, die an der Position enden, an der die X-Regel beginnt und vervollständigt diejenigen Regeln $Z \rightarrow \beta \cdot X\ \gamma$, bei denen auf den Punkt ein X folgt. (Sie gehen in der aktuellen Zeile der Chart soweit nach links wie möglich und suchen dann in der ganzen Spalte.) Der Punkt wird über das X hinwegverschoben, und die neue Regel $Z \rightarrow \beta\ X \cdot \gamma$ wird in derselben Zeile wie die vervollständigte Regel $Z \rightarrow \beta \cdot X\ \gamma$ sowie der aktuellen Spalte eingetragen.

Alle Punktregeln in Zeile i der Chart, haben die Startposition i.

Alle Punktregeln in Spalte j der Chart, haben die Endposition j.

Die Felder (i, i) auf der Diagonale bleiben leer.

	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT –> the .			
1		xxxx			
2			xxxx		
3				xxxx	
4					xxxx

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- The algorithm starts with scanning the first word "the"
  - look at the lexicon for rules that lead to "the"
  - we found DT –> the
  - add this rule to cell 0,1 (from "the" column, go to the bottommost cell)
  - also add a dot after "the"

DT
the

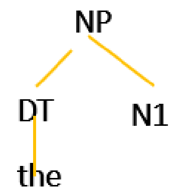
	0	1	2	3	4
		the	**young**	**girl**	**slept**
0	xxxx	DT -> the . NP -> DT . N1			
1		xxxx			
2			xxxx		
3				xxxx	
4					xxxx

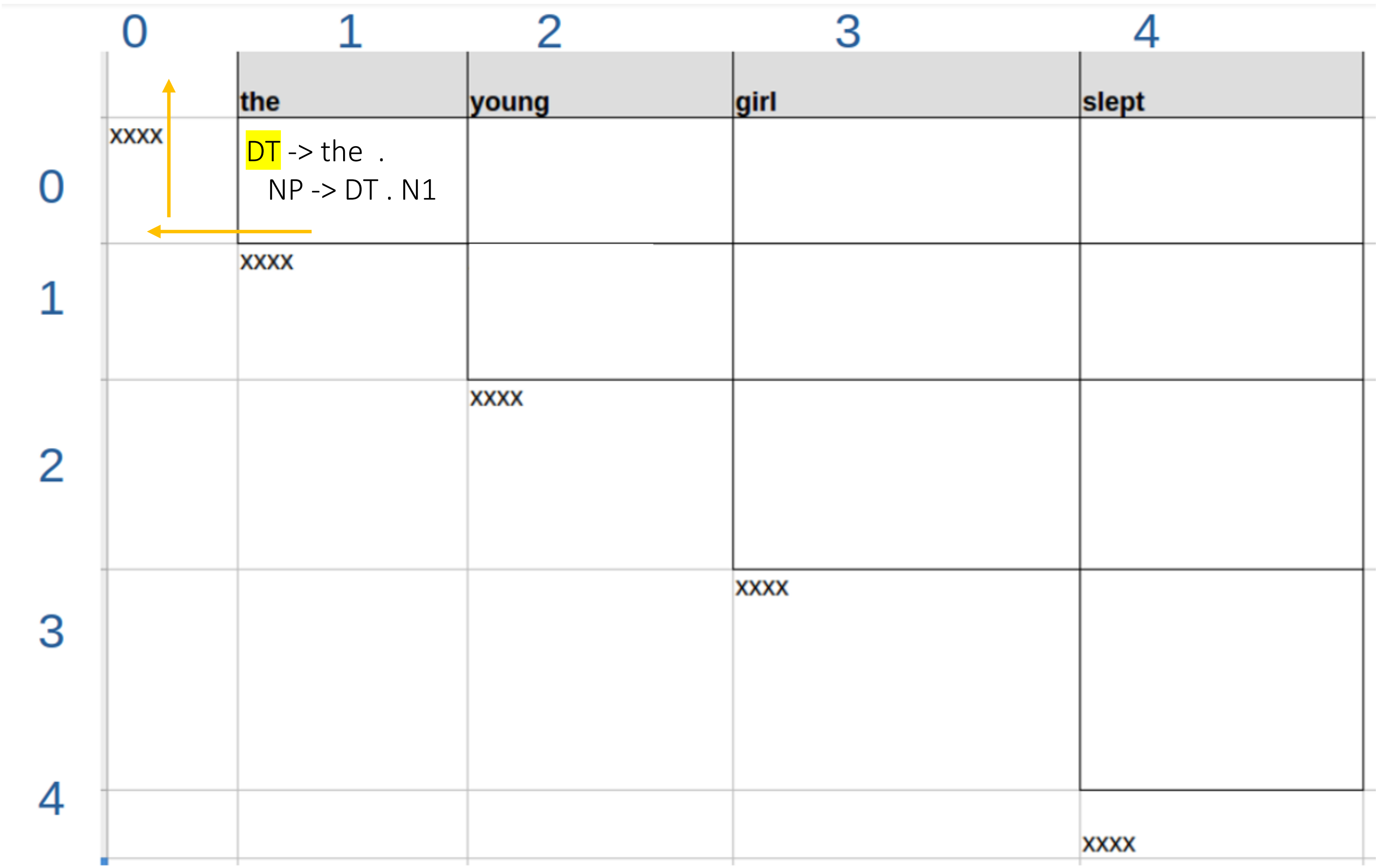
scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

Every time we add a rule to the chart, the algo checks if there is a dot at the end of that rule. If so, it will perform a predict and a complete operation.

- here we can predict DT -> the .
  - predict: look at the grammar, find all rules of the form any -> DT, any (meaning rules, in which DT is the first symbol after ->)
  - found NP -> DT, N1
  - add this rule into the same cell and add a dot after DT. The predict operation is done.
  - next, do the complete operation





scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- marking the symbol with yellow means we are processing it
- if the rule does not have a dot at the end, we do not have to process it (no marking)
- if a rule can be processed (has a dot at the end) and is waiting, mark it with blue.

complete DT -> the .

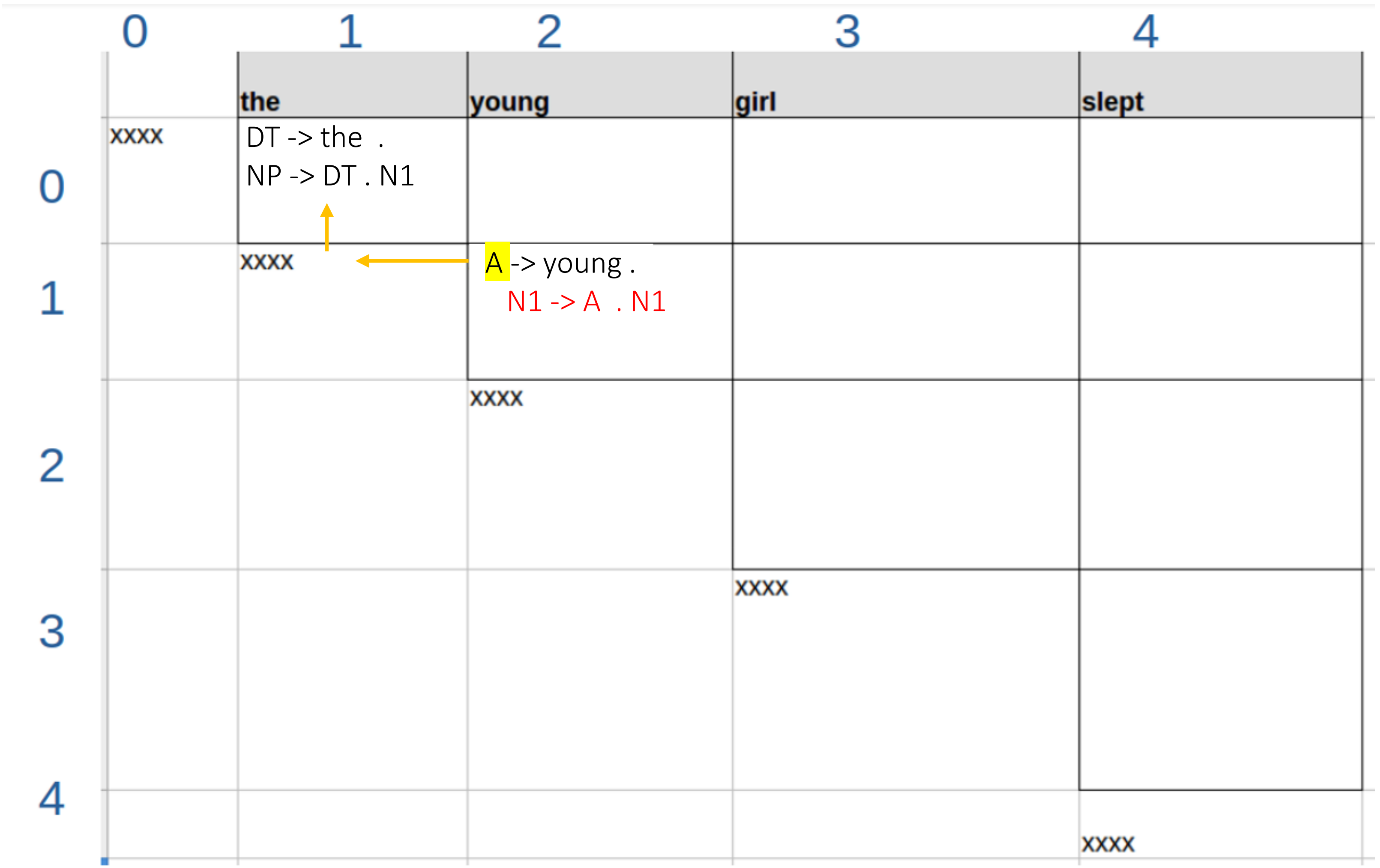
- complete = from the cell where DT -> the . is located, go left until arriving at the cell with XXXX, then go up, check every cell (in the upper rows), look for every rule where DT is after a dot.
- In this case, we can not go up anymore, so we can finish the complete operation.

	0	1	2	3	4
		the	young	girl	slept
0	XXXX	DT -> the . NP -> DT . N1			
1		XXXX	A -> young .		
2			XXXX		
3				XXXX	
4					XXXX

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- We can not expand(or process) any non- terminal symbol anymore, so we scan the next word.



scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

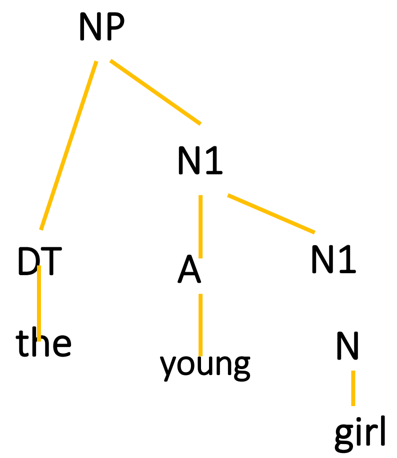
- A -> young . has a dot at the end, so we can predict and complete it
  - predict: look at the grammar, found N1 -> A N1, add it to the cell + add a dot after A
  - complete: go left until XXXX and go up, search for rules with a dot before A, we found no such rule, so we can finish the complete operation.

	0	1	2	3	4
		the	**young**	**girl**	**slept**
0	xxxx	DT -> the . NP -> DT . N1			
1		xxxx	A -> young . N1 -> A . N1		
2			xxxx	N -> girl .	
3				xxxx	
4					xxxx

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- We can not expand any non- terminal symbol anymore, so we scan the next word.
- scan: add N -> girl .

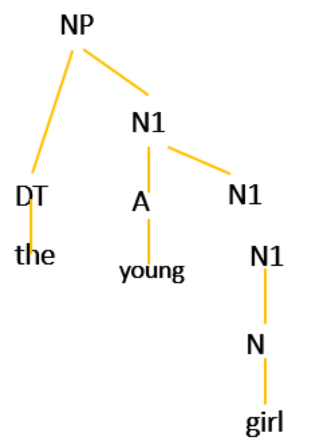


	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT -> the . NP -> DT . N1			
1		xxxx	A -> young . N1 -> A . N1		
2			xxxx complete	N -> girl . N1 -> N .	
3				xxxx	
4					xxxx

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- predict and complete N -> girl .
  - predict: add N1 -> N . (and mark it with blue because we can continue processing it)
  - complete: we can not add anything here
- done processing N -> girl, next we will process N1 -> N .



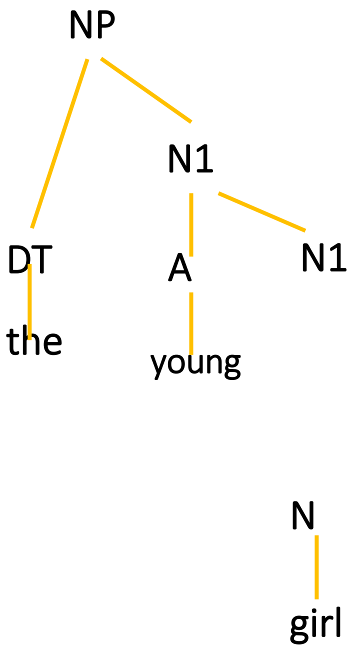
	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT -> the . NP -> DT . N1			
1		xxxx	A -> young . N1 -> A . N1	N1 -> A N1 .	
2			xxxx	N -> girl . N1 -> N . NP -> N1 .	
3				xxxx	
4					xxxx

scan
predict
complete

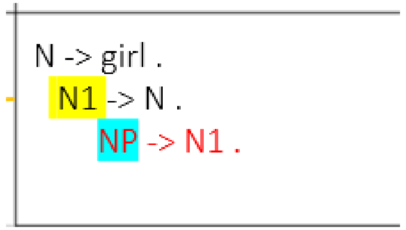
Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

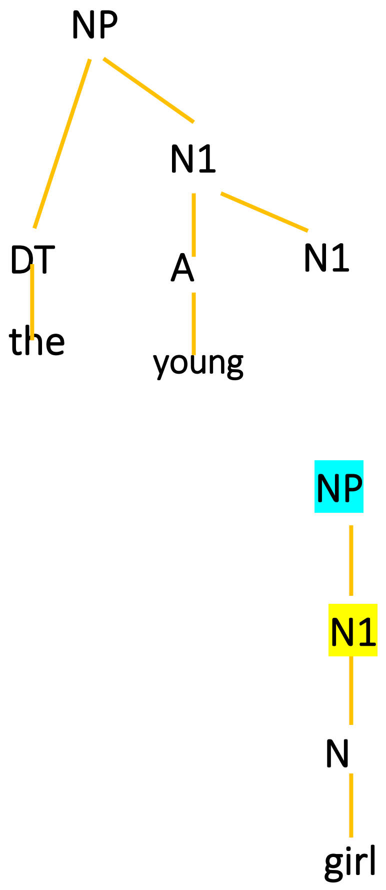
- process N1 -> N .
  - predict: add NP -> N1 . (and mark it with blue because we can continue processing it)
  - complete: found N1 -> A . N1 (dot is before N1), so we copy it to cell 1,3 and also move the dot

no rule has been completet yet



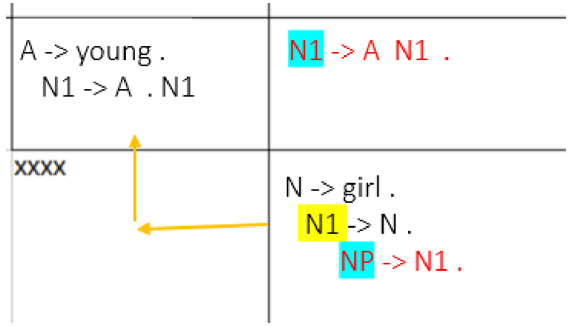
predict N1 -> N

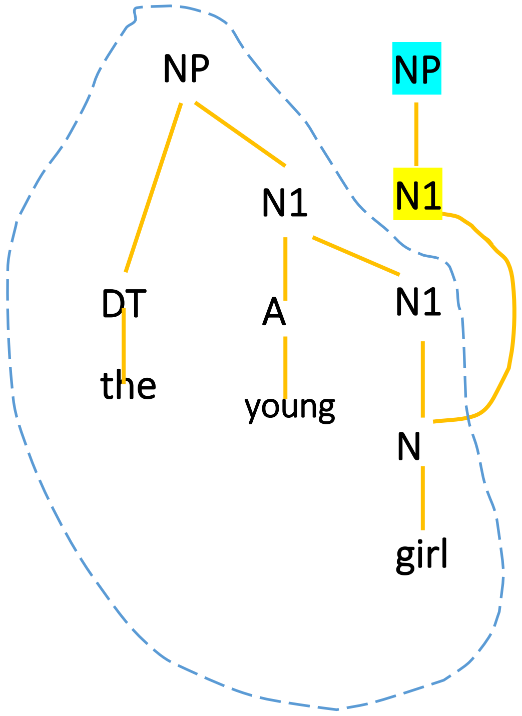




after completing
N1 -> A . N1

We get a complete constituent





	0	1 the	2 young	3 girl	4 slept
0	xxxx	DT -> the . NP -> DT . N1			
1		xxxx	A -> young . N1 -> A . N1	N1 -> A N1 .	
2			xxxx	N -> girl . N1 -> N . NP -> N1 . S -> NP . VP NP -> NP . PP	
3				xxxx	
4					xxxx

scan
predict
complete

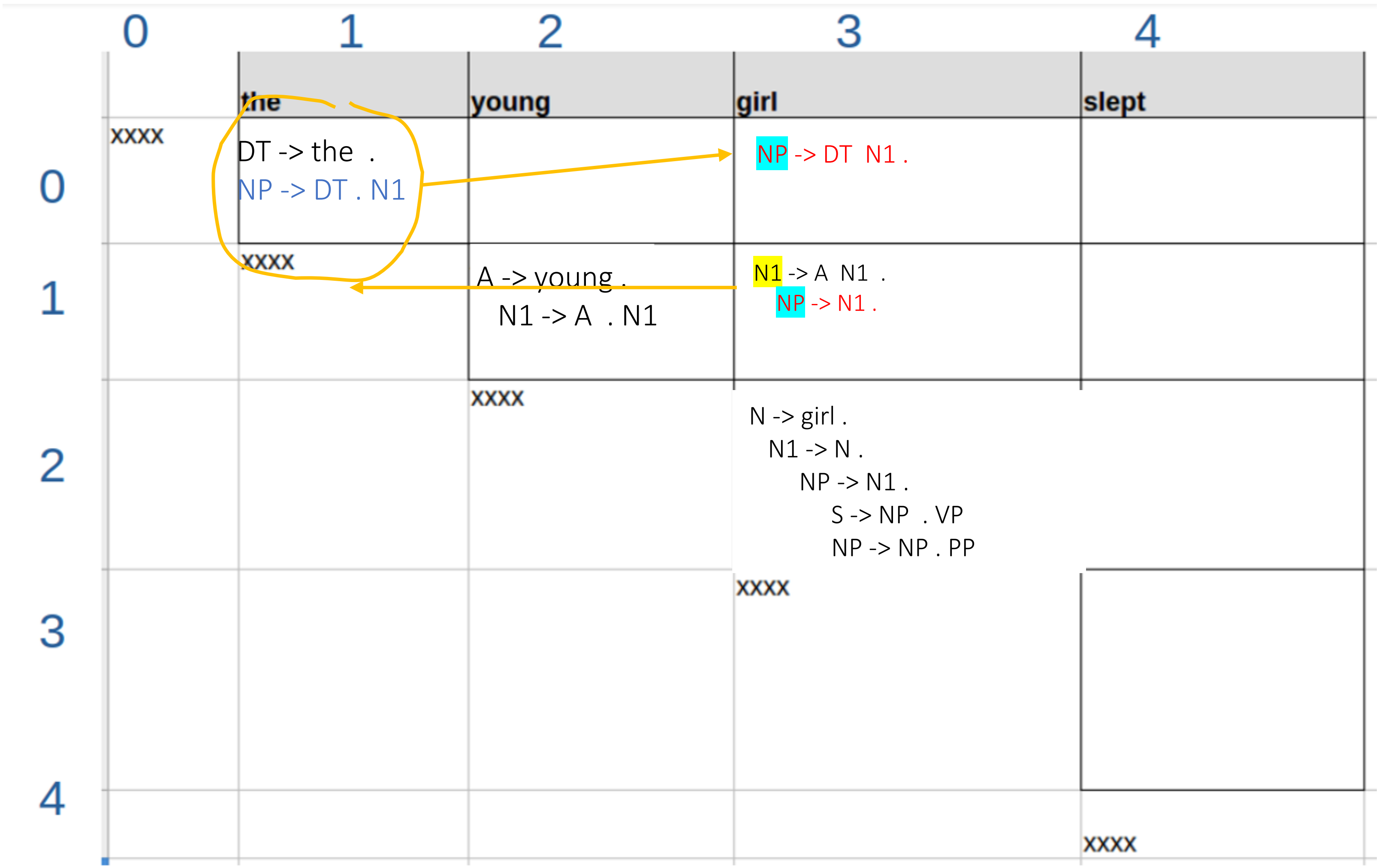
Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- predict and complete NP -> N1 .
  - predict: add S -> NP VP and NP -> NP PP (+ add a dot after NP)
  - complete: nothing to add, we found no rule where a dot is before NP

	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT -> the . NP -> DT . N1			
1		xxxx	A -> young . N1 -> A . N1	N1 -> A N1 .	
2			xxxx	N -> girl . N1 -> N . NP -> N1 . S -> NP . VP NP -> NP . PP	
3				xxxx	
4					xxxx

scan
predict
complete

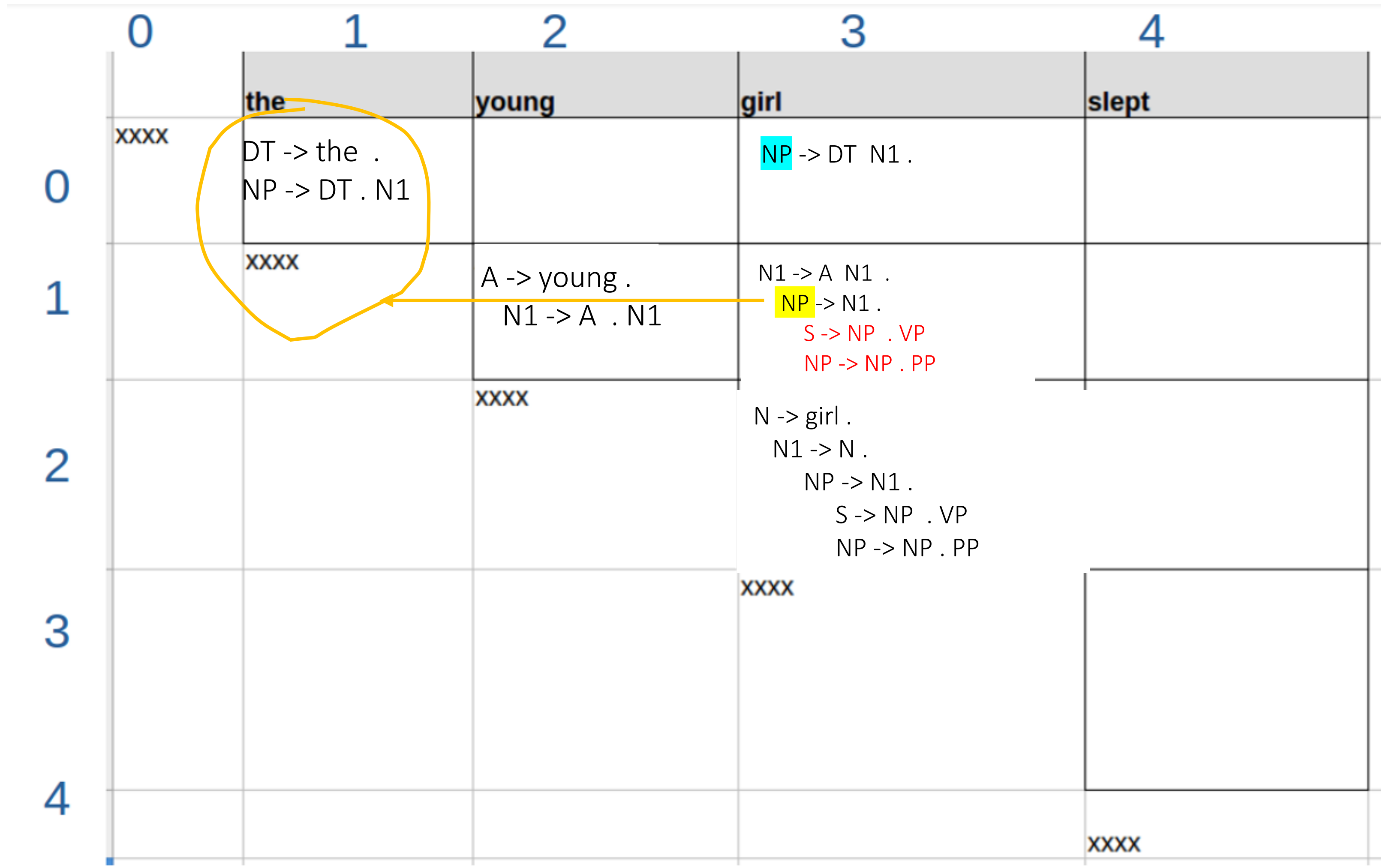
Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with



scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

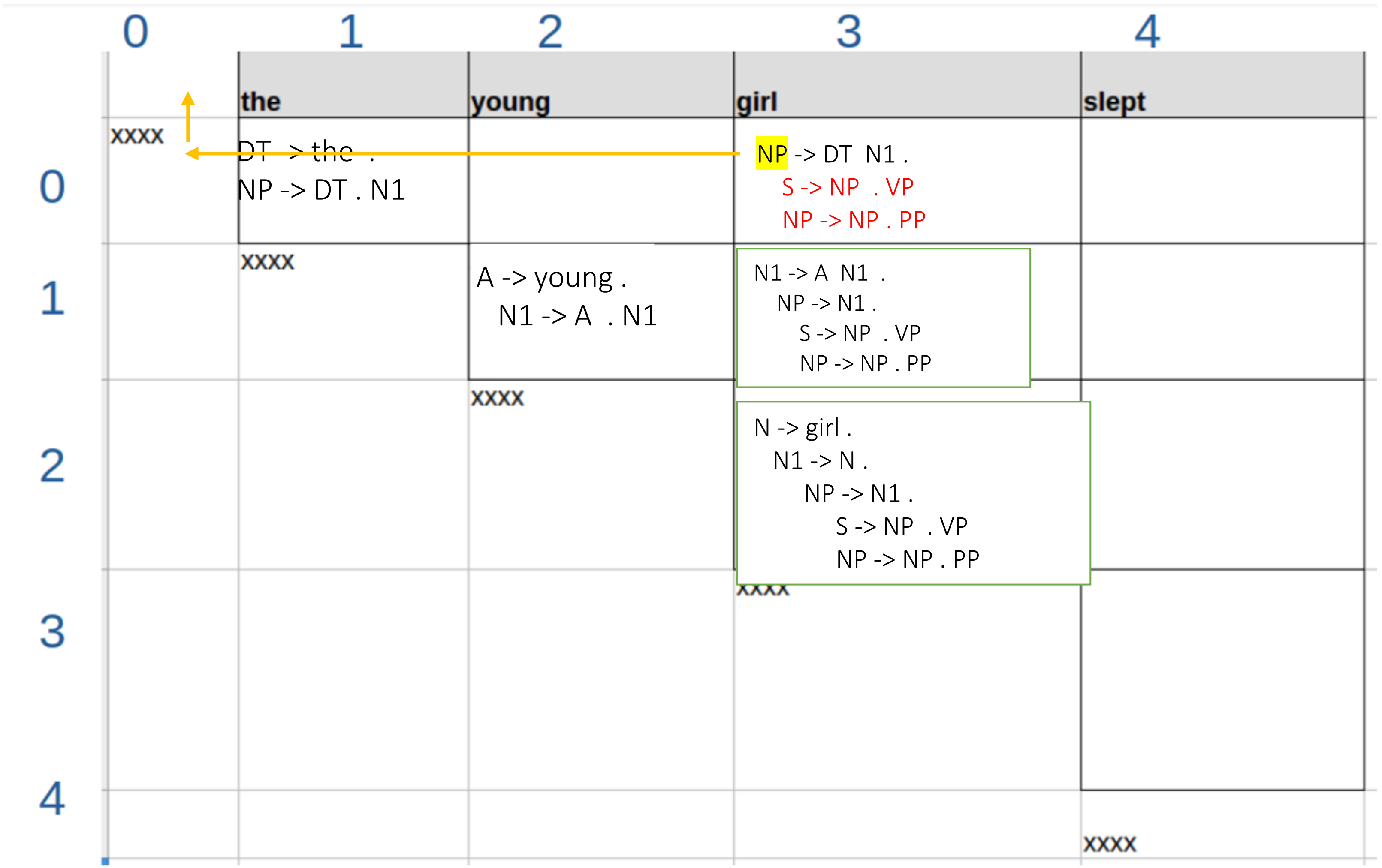
- process N1 -> A N1 .
  - predict: add NP -> N1 .
  - complete: add NP -> DT N1 . to cell 0,3



scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- process NP -> N1 .
  - predict: add S -> NP . VP and NP -> NP . VP
  - complete: nothing



scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- process NP -> DT N1 .
  - predict: add S -> NP . VP and NP -> NP . VP
  - complete: there is no cell that we can look

	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT -> the . NP -> DT . N1		NP -> DT N1 . S -> NP . VP NP -> NP . PP	
1		xxxx	A -> young . N1 -> A . N1	N1 -> A N1 . NP -> N1 . S -> NP . VP NP -> NP . PP	
2			xxxx	N -> girl . N1 -> N . NP -> N1 . S -> NP . VP NP -> NP . PP	
3				xxxx	
4					xxxx

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

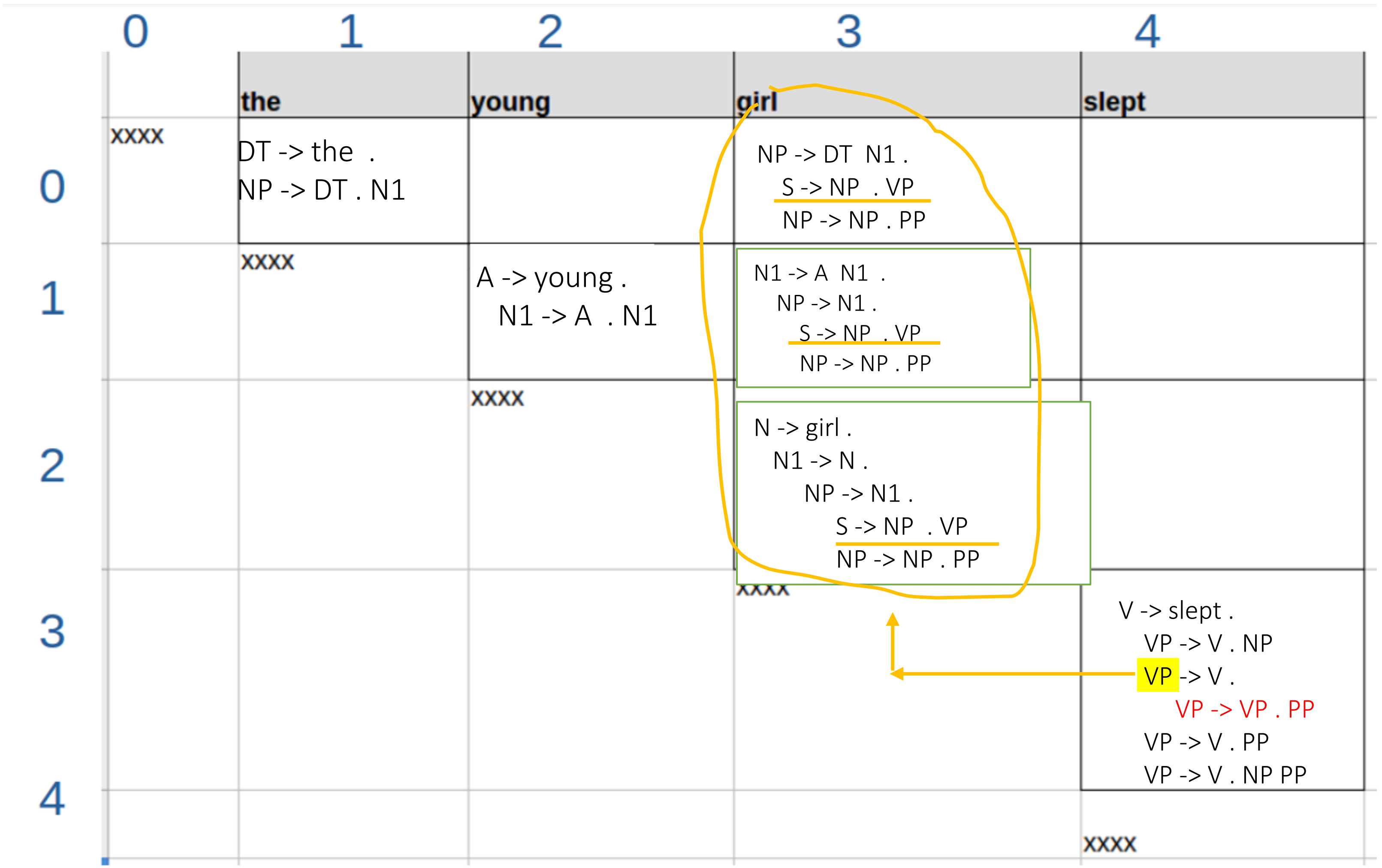
- scan the next word

	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT -> the . NP -> DT . N1		NP -> DT N1 . S -> NP . VP NP -> NP . PP	
1		xxxx	A -> young . N1 -> A . N1	N1 -> A N1 . NP -> N1 . S -> NP . VP NP -> NP . PP	
2			xxxx	N -> girl . N1 -> N . NP -> N1 . S -> NP . VP NP -> NP . PP	
3				xxxx	V -> slept . VP -> V . NP VP -> V . VP -> V . PP VP -> V . NP PP
4					xxxx

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- scan: add V -> slept .
  - predict: add VP -> V, NP and all other rules where V is after ->
  - complete: look for rules where V is after a dot. We found no such rule here, so we are finished.



scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- process VP -> V .
  - predict: add VP -> VP . PP
  - complete: look for all rules (in column 3) where a dot is before VP, copy those rules to column 4 and move the dot.

	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT -> the . NP -> DT . N1		NP -> DT N1 . S -> NP . VP NP -> NP . PP	S -> NP VP .
1		xxxx	A -> young . N1 -> A . N1	N1 -> A N1 . NP -> N1 . S -> NP . VP NP -> NP . PP	S -> NP VP .
2			xxxx	N -> girl . N1 -> N . NP -> N1 . S -> NP . VP NP -> NP . PP	S -> NP VP .
3				xxxx	V -> slept . VP -> V . NP VP -> V . VP -> VP . PP VP -> V . PP VP -> V . NP PP
4					xxxx

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

	0	1	2	3	4
		the	young	girl	slept
0	xxxx	DT -> the . NP -> DT . N1		NP -> DT N1 . S -> NP . VP NP -> NP . PP	S -> NP VP .
1		xxxx	A -> young . N1 -> A . N1	N1 -> A N1 . NP -> N1 . S -> NP . VP NP -> NP . PP	S -> NP VP .
2			xxxx	N -> girl . N1 -> N . NP -> N1 . S -> NP . VP NP -> NP . PP	S -> NP VP .
3				xxxx	V -> slept . VP -> V . NP VP -> V . VP -> VP . PP VP -> V . PP VP -> V . NP PP
4					xxxx

scan
predict
complete

Beispielgrammatik	Beispiellexikon
S NP VP	0.6 DT the
VP VP PP	0.4 DT a
VP V NP	0.2 A old
VP V	0.3 A young
VP V PP	0.2 A big
VP V NP PP	0.3 A small
NP NP PP	0.2 N man
NP DT N1	0.3 N hill
NP N1	0.2 N telescope
N1 A N1	0.2 N girl
N1 N	0.1 N saw
PP P NP	0.4 V saw
	0.6 V slept
	0.6 P on
	0.4 P with

- expand the rest (= process the yellow and blue rules)
- the result is that no more new rules will be added to the chart, and the algorithm will terminate.
- check if we have at least one **S rule at the top-left most cell.** In this case, we have.
- This means the sentence is successfully parsed and is a grammatical sentence.

done

Parsewald

- We can retrieve the Parsewald by tracing backward, starting from S.
- This means when we add a new rule, would first have to store information about which rule/symbol the current rule is expanded from.

Regarding Viterbi

In Übung 10 : Alle Regeln haben eine Wahrscheinlichkeit(log-prob). Bei der Durchführung des Algorithmus wird die Viterbi-Maximierung auch gemacht (If the rule we want to add already exists in the cell, then we choose the rule that has the higher probablity).

	the	young	girl	slept
xxxx	DT → the . log(0.6) NP → DT . N1 log(0.6) + log(0.5)= -1.2 (predict)		NP → DT N1 . -1,2 + 0.0463 complete S → NP . VP NP → NP . PP	S → NP VP .
	xxxx	A → young . N1 → A . N1	N1 → A N1 . 0.0463 NP → N1 . S → NP . VP NP → NP . PP	S → NP VP .
		xxxx	N → girl . N1 → N . NP → N1 . S → NP . VP NP → NP . PP	S → NP VP .
			xxxx	V → slept . VP → V . NP VP → V. VP → V . PP VP → V . NP PP VP → VP . PP
				xxxx

NP → DT . N1 is predicted by DT → the .
The prob of DT → the . is
the prob of DT → the (from the grammar)
+ the prob of NP → DT . N1

The probs of rules that are added by the complete operation are calculated the same way.

```
0.6 DT the
0.4 DT a
0.2 A old
0.3 A young
0.2 A big
0.3 A small
```

```
1.0 S NP VP
0.2 VP VP PP
0.3 VP V NP
0.2 VP V
0.2 VP V PP
0.1 VP V NP PP
0.2 NP NP PP
0.5 NP DT N1
0.3 NP N1
0.3 N1 A N1
0.7 N1 N
1.0 PP P NP
```

note: fake numbers are used in the example