Scan Trägt an Position i,i+1 alle Regeln der Form $\underline{X \to w_i \cdot}$ ein (wobei w_i das i-te Wort und $X \to w_i$ eine Grammatikregel ist).

Predict Wenn eine Punktregel $\underline{X \to \alpha \cdot}$ an Position i,k in die Chart eingetragen wird, dann trägt die Predict-Operation alle Regeln der Form $\underline{Y \to X \cdot \beta}$ ebenfalls an Position i,k ein (wobei X und Y beliebige Nichtterminale und α, β beliebige Folgen von Terminalen und Nichtterminalen sind.)

Complete Wenn eine Punktregel $\underline{X \to \alpha \cdot}$ an Position j,k in die Chart eingetragen wird, dann such die Complete-Operation alle Regeln der Form $\underline{Y \to \alpha \cdot X \ \beta}$ in i,j (i beliebig) und trägt die neue Regel $\underline{Y \to \alpha \ X \cdot \beta}$ an Position i,k ein.

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

Child rule

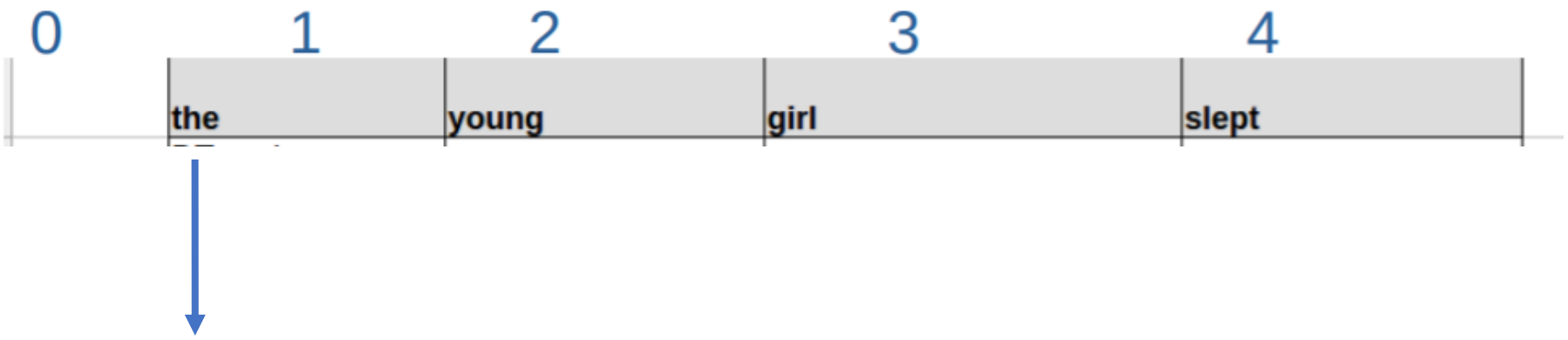
	the	young	girl	slept
xxxx	DT → the . log(0.6) NP → DT . N1 og(0.6) + log(0.5)= -1.2 (predict)		NP → DT N1 . -1,2 + 0.0463 complete S → NP . VP NP → NP . PP	S → NP VP .
	xxxx	A → young . N1 → A . N1	N1 → A N1 . 0.0463 NP → N1 . S → NP . VP NP → NP . PP	S → NP VP .
		xxxx	N → girl . N1 → N . NP → N1 .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

```
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
```

Each rule stores its previous rule(or called “child” because we are doing a bottom up parsing).
For example, DT → the . has no child rule, NP → DT . N1 has DT → the . as child rule
This allows us to backtract to get the analysis.

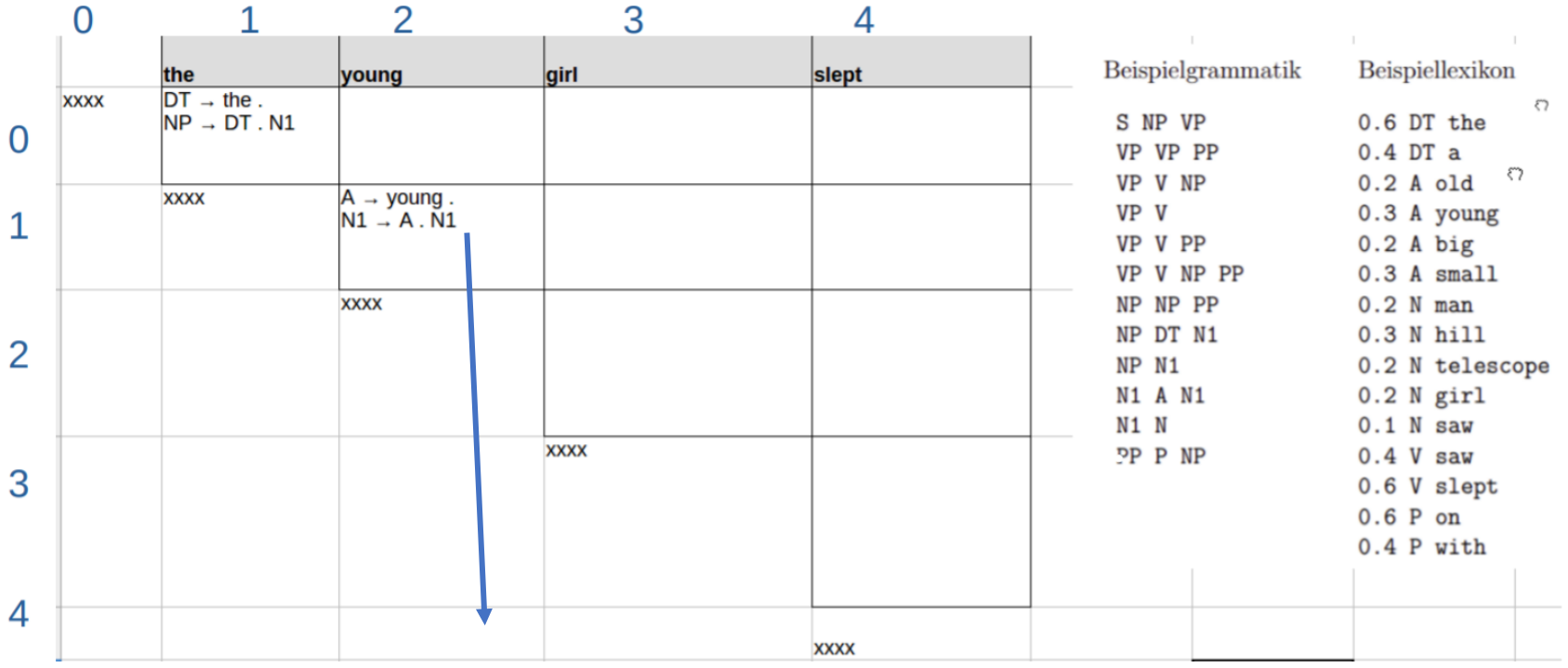
Explain start and end position



„the“ starts at position 0 and ends at position 1

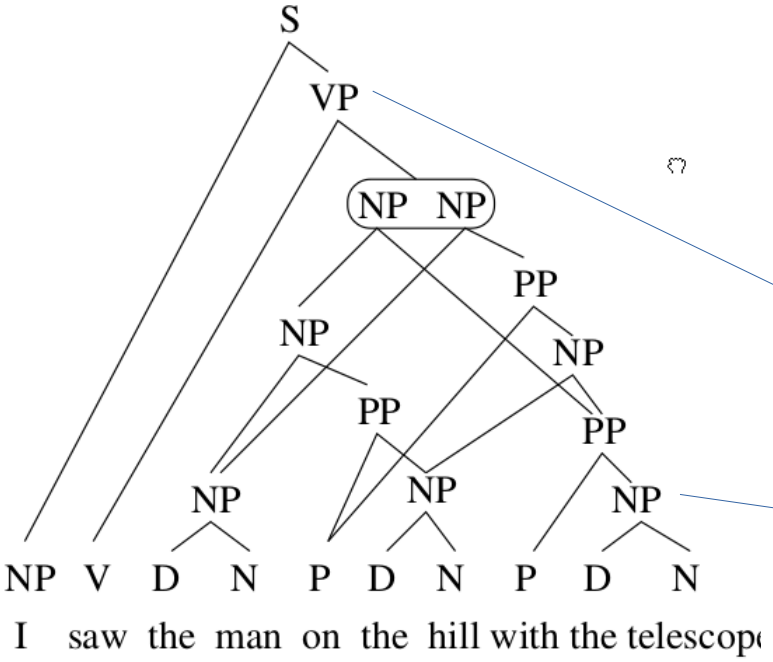
„young“ starts at position 1 and ends at position 2

In the chart, column numbers represent **the end positions** of each word and the rule related to that word. Row numbers represent **the start positions.**



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

Rules in this cell have star position = 1 and end position = 2 meaning, these rule cover the words that starts at position 1 and ends at 2. In this case, the word “young”. For example, N1 → A . N1 means the parser already read A (because the dot is after A) and A is a category (symbol) that can expand to “young”



compare start/end position to the parse tree

We can see that each symbol will cover a certain spand of the tokens.

VP → V NP covers position 1 to 10 (“saw” til “telescope”)

this NP → D N covers “the telescope”

How can we represent the grammar and lexicon?

```
# Grammatik oder Lexikon einlesen
# logruleprobs['XP'] liefert alle Regeln mit XP als erstem Element
# auf der rechten Seite
def read_file(filename):
    logruleprobs = defaultdict(lambda: dict())
    with open(filename) as file:
        for line in file:
            p, lhs, *rhs = line.split()
            logruleprobs[rhs[0]][(lhs,tuple(rhs))] = log(float(p))
    return logruleprobs
```

grammar.txt

```
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
```

```
NP : {('S', ('NP', 'VP')): 0.0, ('NP', ('NP', 'PP')): -1.6094379124341003}
VP : {('VP', ('VP', 'PP')): -1.6094379124341003}
V : {('VP', ('V', 'NP')): -1.2039728043259361, ('VP', ('V',)): -1.609437912
DT : {('NP', ('DT', 'N1')): -0.6931471805599453}
N1 : {('NP', ('N1',)): -1.2039728043259361}
A : {('N1', ('A', 'N1')): -1.2039728043259361}
N : {('N1', ('N',)): -0.35667494393873245}
P : {('PP', ('P', 'NP')): 0.0}
```

How can we represent the grammar and lexicon?

```
# Grammatik oder Lexikon einlesen
# logruleprobs['XP'] liefert alle Regeln mit XP als erstem Element
# auf der rechten Seite
def read_file(filename):
    logruleprobs = defaultdict(lambda: dict())
    with open(filename) as file:
        for line in file:
            p, lhs, *rhs = line.split()
            logruleprobs[rhs[0]][(lhs,tuple(rhs))] = log(float(p))
    return logruleprobs
```

```
NP : {('S', ('NP', 'VP')): 0.0, ('NP', ('NP', 'PP')): -1.6094379124341003}
VP : {('VP', ('VP', 'PP')): -1.6094379124341003}
```

grammar.txt

```
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
```

The algorithm has to read the grammar in scan and predict operations. Assuming we want to predict a symbol or scan a word, the predict and scan operations will have search in the grammar for rules whose first symbol on the rhs matches this given symbol.

This datastructure allows an easy access to all of those rules. For example,
If we want to predict “NP” we can call logruleprobs[“NP”] and it would return the dictionary of all rules with have “NP” on the rhs and their probs.

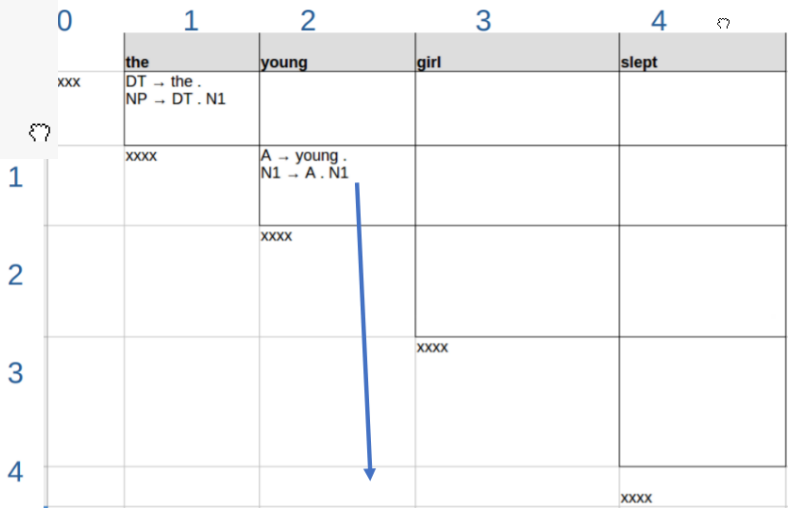
How do we represent the chart in the code?

```
def parse(self, tokens):
    ''' implementiert den Viterbi-Algorithmus für Left-Corner-Parsing '''
    # Datenstrukturen initialisieren
    self.logvitprob = [{} for _ in range(len(tokens)+1)]
```

dot pos

start pos

```
[{},
 {('DT', ('the',), 1, 0): -0.5, ('NP', ('DT', 'N1'), 1, 0): -1.2},
 {('A', ('young',), 1, 1): -1.2, ('N1', ('A', 'N1'), 1, 1): -2.4},
 ...
]
```



We implement the chart as a list of dictionaries called logvitprob. The index of the list represents the column number (end position). Each dictionary stores rules, dot position, start position (row number) in tuples as key and the prob as value. Tuple is called “item” in the code.

Übung: How can you add (DT → the .) with prob -0.5 to cell 0,1 (row, col) if logvitprob is already initiated?
How can you add (N1 → A . N1) with prob 0.1 to cell 1,2 ?

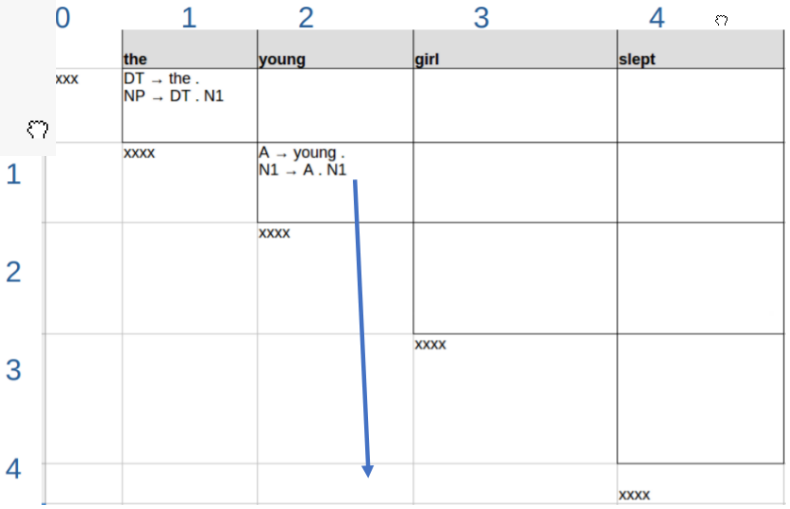
Übung: how can you add (DT → the .) with prob -0.5 to cell 0,1 (row, col)if logvitprob is already initiated?
how can you add (N1 → A N1 .) with prob 0.1 to cell 1,3 if logvitprob is already initiated?

```
def parse(self, tokens):
    ''' implementiert den Viterbi-Algorithmus für Left-Corner-Parsing '''
    # Datenstrukturen initialisieren
    self.logvitprob = [{} for _ in range(len(tokens)+1)]
```

dot pos

start pos

```
[{},
 {('DT', ('the',), 1, 0): -0.5, ('NP', ('DT', 'N1'), 1, 0): -1.2},
 {('A', ('young',), 1, 1): -1.2, ('N1', ('A', 'N1'), 1, 1): -2.4},
 ...
]
```



Answer:

endpos = 1
startpos = 0
dotpos =1
item = ('DT', ('the',), dotpos, startpos)
logvitprob[endpos][item] = -0.5

endpos = 3
startpos = 1
dotpos =2
item = ('N1', ('A','N1'), dotpos, startpos)
logvitprob[endpos][item] = 0.1

```
def parse(self, tokens):
    ''' implementiert den Viterbi-Algorithmus für Left-Corner-Parsing '''
    # Datenstrukturen initialisieren
    self.logvitprob = [{} for _ in range(len(tokens)+1)]
    self.childitem  = [{} for _ in range(len(tokens)+1)]
    # Scan-Operation für jedes Token aufrufen
    for i in range(len(tokens)):
        self.scan(tokens[i], i)

    # beste vollständige S-Konstituente suchen, die den ganzen Satz abdeckt
    bestscore, bestitem = -1e300, None
    for item, score in self.logvitprob[-1].items():
        lhs, rhs, dotpos, startpos = item
        # Hat die Punktregel alle gewünschten Eigenschaften?
        if lhs == 'S' and dotpos == len(rhs) and startpos == 0:
            # Ist die neue Analyse besser als alle bisherigen?
            if bestscore < score:
                bestscore = score
                bestitem = item
    if bestitem is None:  # Fehlermeldung ausgeben
        print('no analysis for:', ' '.join(tokens))
    else: # Parsebaum ausgeben
        self.print_parse(bestitem, len(tokens))
        print('')
```

This function takes tokens as argument and iterates over the tokens, and calls scan function on each token. Scan will call predict and complete recursively until the chart is filled.

Check the most upper right cell if there is any rule of the form S → XX . (S rule with dot at the end).
There can be many S rules in the cell, so the rule with the highest prob will be selected.

The function print_parse will print the selected S rule and its child rules recusively until the last child.

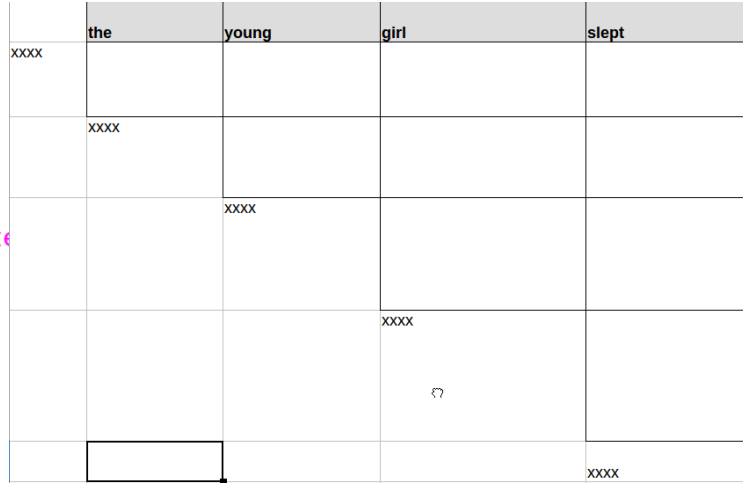
```
Best Analysis
(S (NP (DT the)(N1 (A young)(N1 (N girl))))(VP (V slept)))
```

```
def add(self, lhs, rhs, dotpos, startpos, endpos, logprob, child):
    ''' Trägt eine Punktregel in die Chart ein '''

    item = (lhs, rhs, dotpos, startpos)
    # Viterbi-Maximierung
    if item not in self.logvitprob[endpos] or self.logvitprob[endpos][item] < logprob:
        self.logvitprob[endpos][item] = logprob
        self.childitem[endpos][item] = child  # Verweis auf Tochterkonstitu
        if dotpos == len(rhs):
            # Konstituente vollständig erkannt
            self.predict(lhs, startpos, endpos, logprob, item)
            self.complete(lhs, startpos, endpos, logprob, item)

def scan(self, token, pos):
    ''' Scannt das nächste Token '''
    if token not in self.loglexprobs:
        # Das Wort ist nicht im Lexikon
        print('unknown word:', token, file=sys.stderr)
    else:
        for (lhs, rhs), logp in self.loglexprobs[token].items():
            # Alle möglichen Wortarten aus dem Lexikon eintragen
            self.add(lhs, rhs, 1, pos, pos+1, logp, None);

def predict(self, cat, startpos, endpos, logprob, child):
    ''' trägt alle Regeln ein, deren rechte Seite mit cat beginnt '''
    for (lhs, rhs), logp in self.logruleprobs[cat].items():
        self.add(lhs, rhs, 1, startpos, endpos, logprob+logp, child)

def complete(self, cat, splitpos, endpos, logprob, child):
    ''' vervollständigt Punktregeln, die die gerade gefundene Konstituente erwarte
    # Alle Punktregeln durchlaufen, die an splitpos enden
    for item, logp in self.logvitprob[splitpos].items():
        lhs, rhs, dotpos, startpos = item
        # Test, ob die Punktregel mit der als Argument
        # übergebenen Punktregel vervollständigt werden kann
        if dotpos < len(rhs) and rhs[dotpos] == cat:
            self.add(lhs, rhs, dotpos+1, startpos, endpos, logp+logprob, child)
```

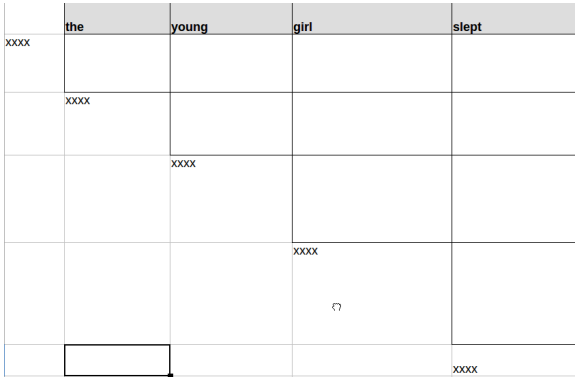
We have 4 main functions that will be used to run the algorithm.

add: add a given rule to the given cell position, then check if this rule can be predicted and completed or not. If a rule has a dot at the end, we can predict the category(lhs) of it. E.g. for “**NP** → N PP .”, we will predict NP.

scan: get a word token as input and check if it can be scanned or not. If so, then call the “add” function to add it to the suitable cell.



```python
def add(self, lhs, rhs, dotpos, startpos, endpos, logprob, child):
    ''' Trägt eine Punktregel in die Chart ein '''

    item = (lhs, rhs, dotpos, startpos)
    # Viterbi-Maximierung
    if item not in self.logvitprob[endpos] or self.logvitprob[endpos][item] < logprob:
        self.logvitprob[endpos][item] = logprob
        self.childitem[endpos][item] = child  # Verweis auf Tochterkonstituente
        if dotpos == len(rhs):
            # Konstituente vollständig erkannt
            self.predict(lhs, startpos, endpos, logprob, item)
            self.complete(lhs, startpos, endpos, logprob, item)
```




```python
def scan(self, token, pos):
    ''' Scannt das nächste Token '''
    if token not in self.loglexprobs:
        # Das Wort ist nicht im Lexikon
        print('unknown word:', token, file=sys.stderr)
    else:
        for (lhs, rhs), logp in self.loglexprobs[token].items():
            # Alle möglichen Wortarten aus dem Lexikon eintragen
            self.add(lhs, rhs, 1, pos, pos+1, logp, None);

def predict(self, cat, startpos, endpos, logprob, child):
    ''' trägt alle Regeln ein, deren rechte Seite mit cat beginnt '''
    for (lhs, rhs), logp in self.logruleprobs[cat].items():
        self.add(lhs, rhs, 1, startpos, endpos, logprob+logp, child)

def complete(self, cat, splitpos, endpos, logprob, child):
    ''' vervollständigt Punktregeln, die die gerade gefundene Konstit
    # Alle Punktregeln durchlaufen, die an splitpos enden
    for item, logp in self.logvitprob[splitpos].items():
        lhs, rhs, dotpos, startpos = item
        # Test, ob die Punktregel mit der als Argument
        # übergebenen Punktregel vervollständigt werden kann
        if dotpos < len(rhs) and rhs[dotpos] == cat:
            self.add(lhs, rhs, dotpos+1, startpos, endpos, logp+logpr
```

predict: The predict function gets a category symbol (e.g. NP) as input and predicts this symbol by looking at the grammar and add the relevant rules to the corresponding cell (again using “add” function).

complete: this function gets a category symbol (e.g. N from cell with start=2, end=3) as input and looks at all rules in the column(endpos) = 2 for rules that has a dot before the symnol N (such as NP → D **. N**). It copies these rules to the cell with start=3, end = 2 (again using “add” function).

After the add function added a given rule to the chart it will call predict and complete on this rule recursively until all rules are processed. Then program will scan the next word.

Übung: Analyse the sentence “the young man slept on the hill” with left-corner parser. Try to compute like the code would do. Then extract the best analysis and draw a tree.

Check your answer in :
https://colab.research.google.com/drive/1ctNQSEYSBlPE6KRDK0UL2uG7CIvlJ7i3?usp=sharing

To prepare for the Übungklausur, I suggest you try to write the code 1-2 times without looking at the solution. Try to think of when you fill the chart manually when you write each function. Each line that you write should make sense to you (you know what the line does, how each variable look like, and which step in the algorihm are you at). If it does not then try to get a better understanding of the code.