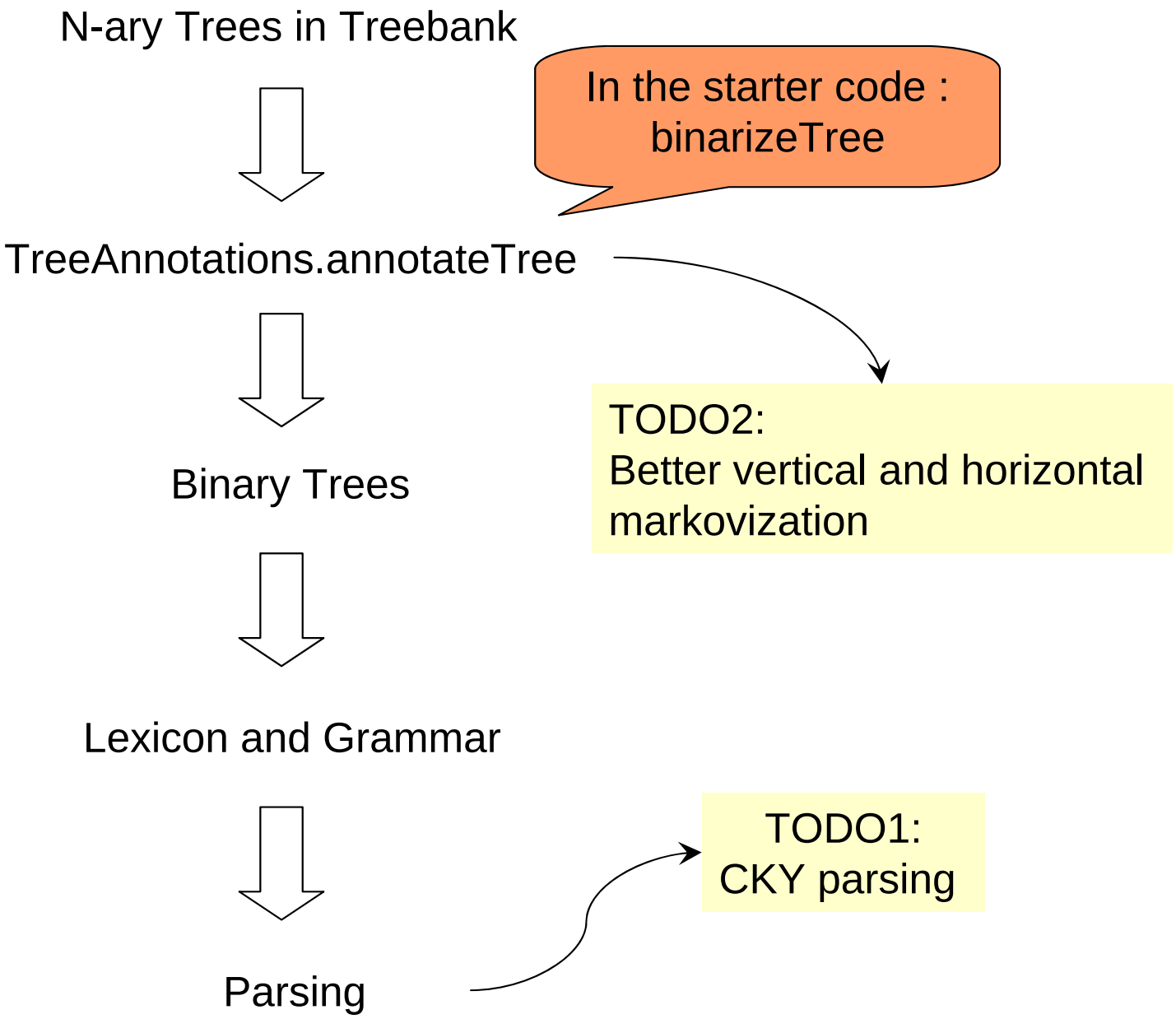
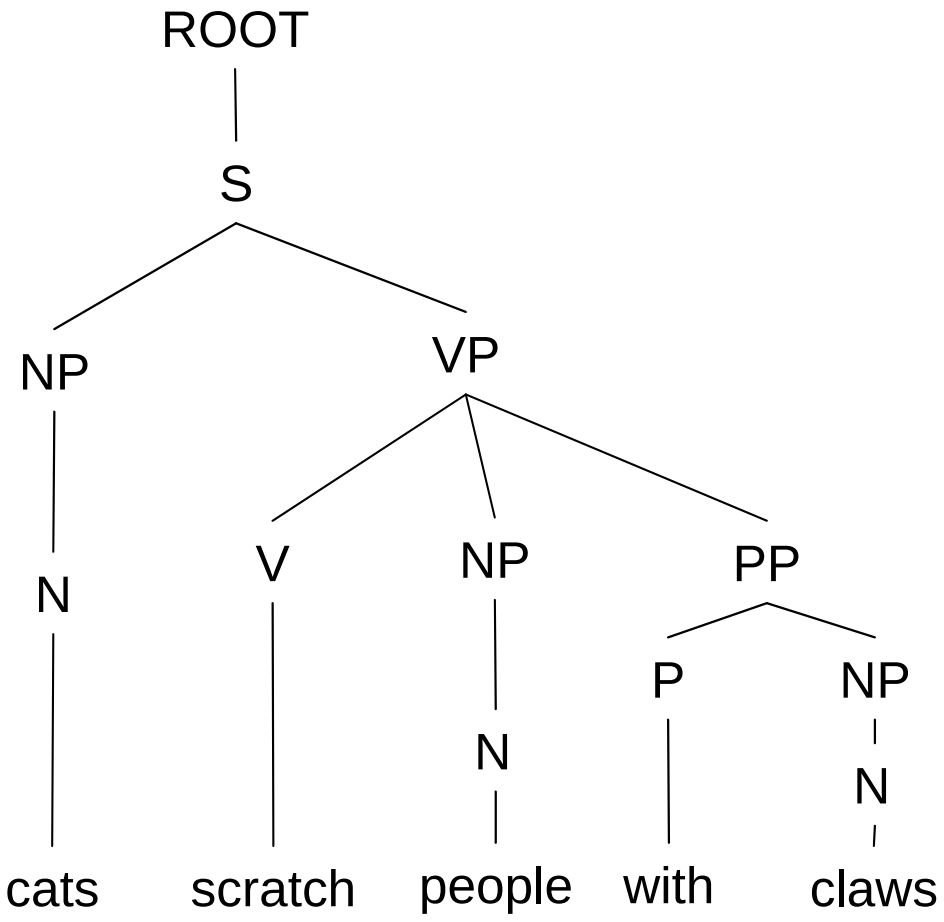
Section 5: Parsing & PCFGs

May 12, 2006

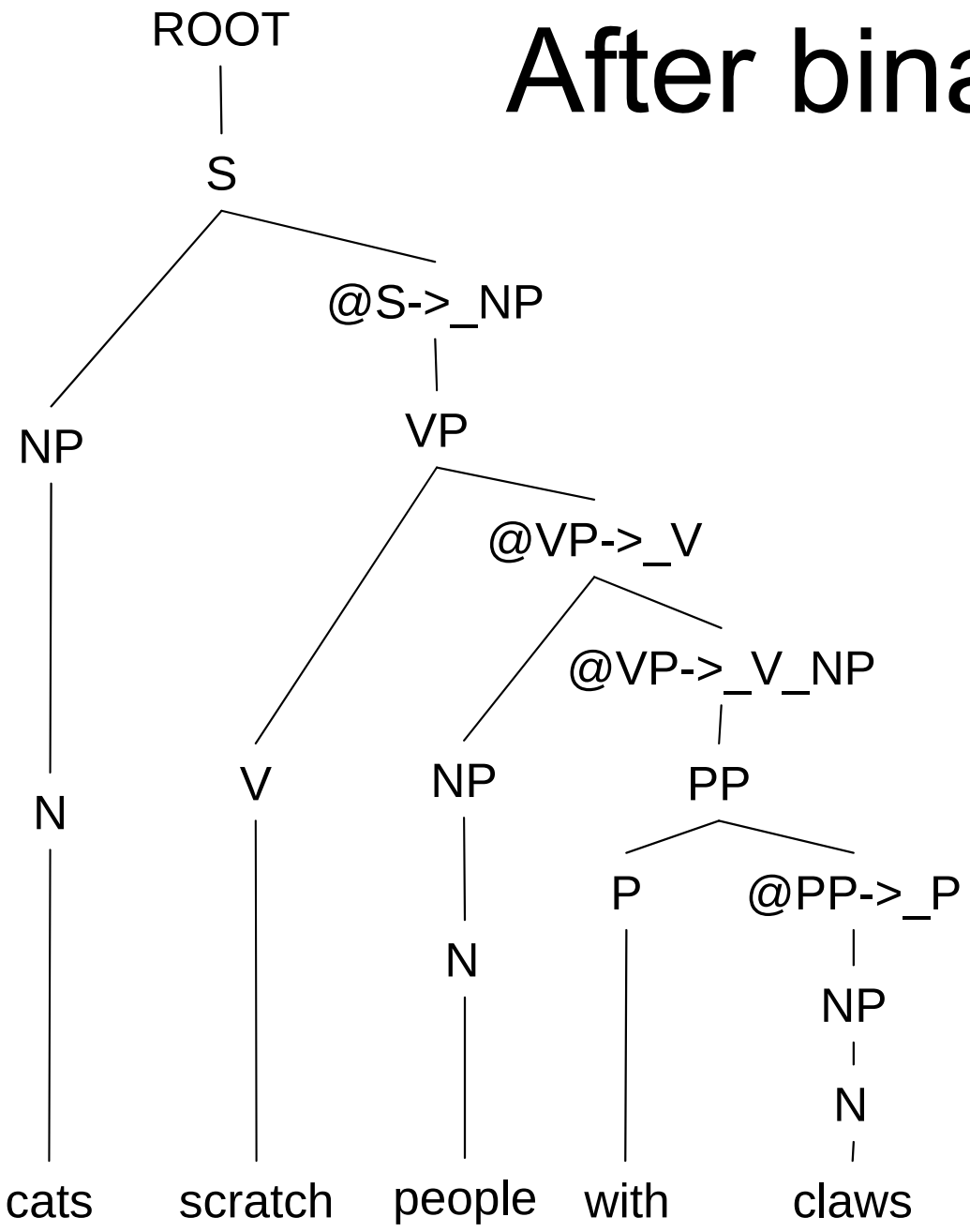
Pi-Chuan Chang

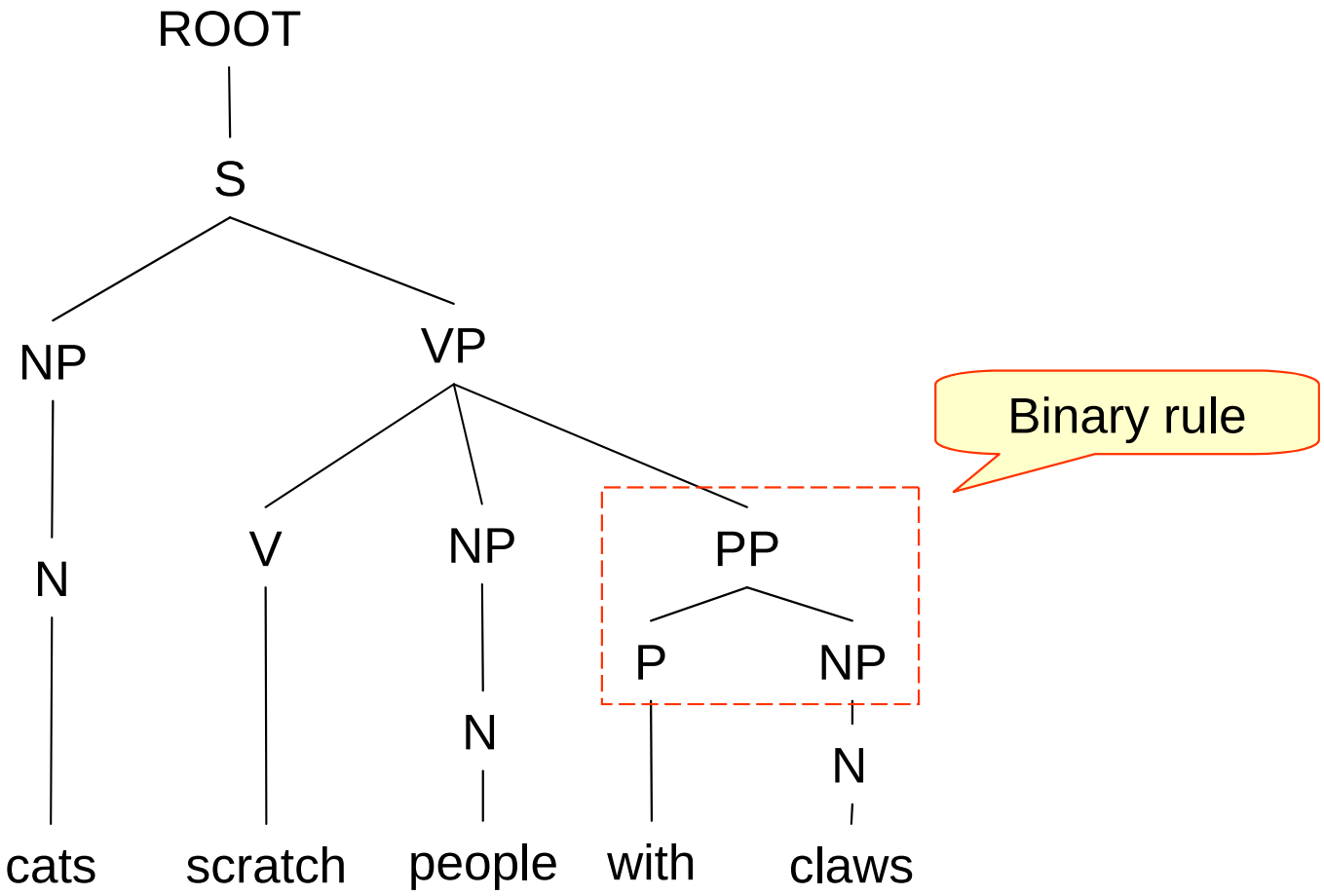

N-ary Trees in Treebank
In the starter code :
binarizeTree
TreeAnnotations.annotateTree
TODO2:
Better vertical and horizontal
markovization
Binary Trees
Lexicon and Grammar
TODO1:
CKY parsing
Parsing

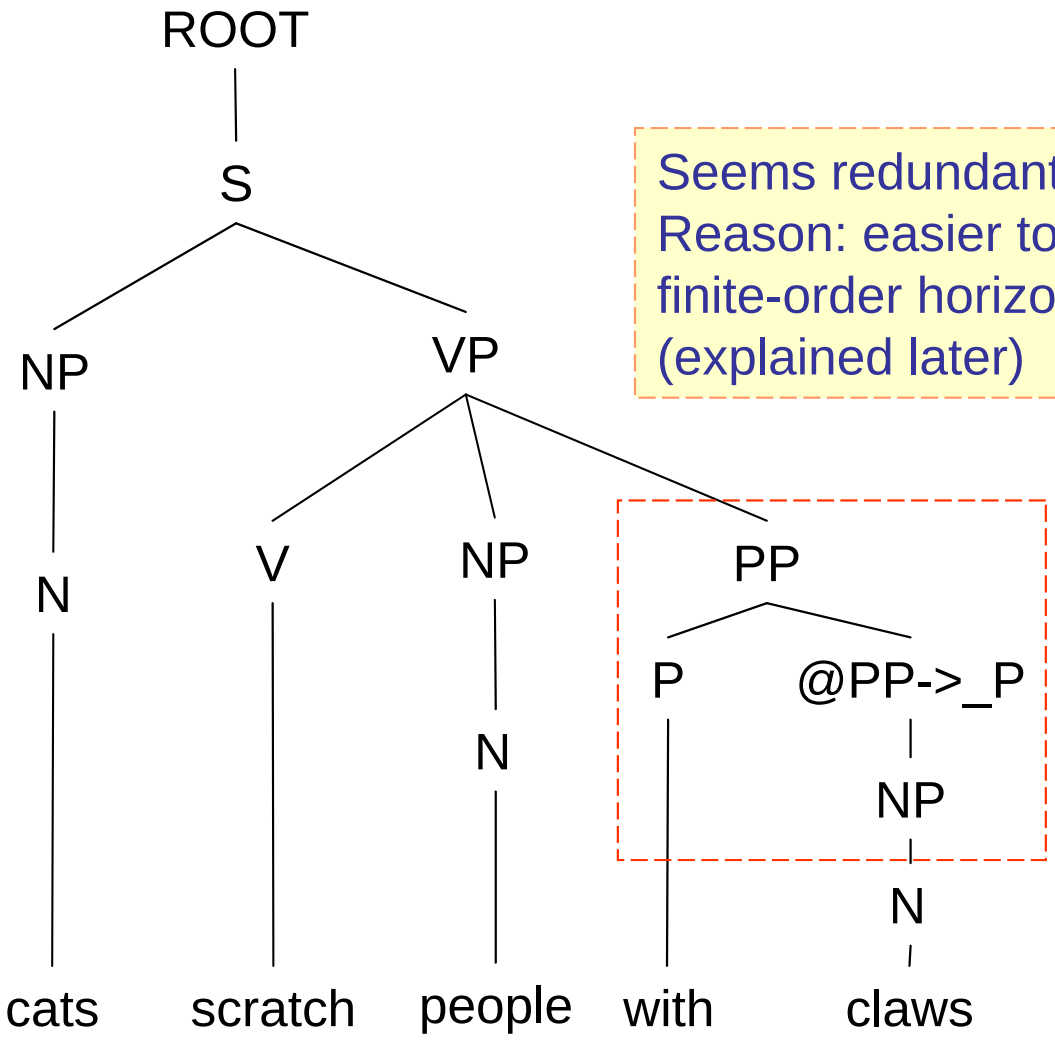
An example: before binarization…



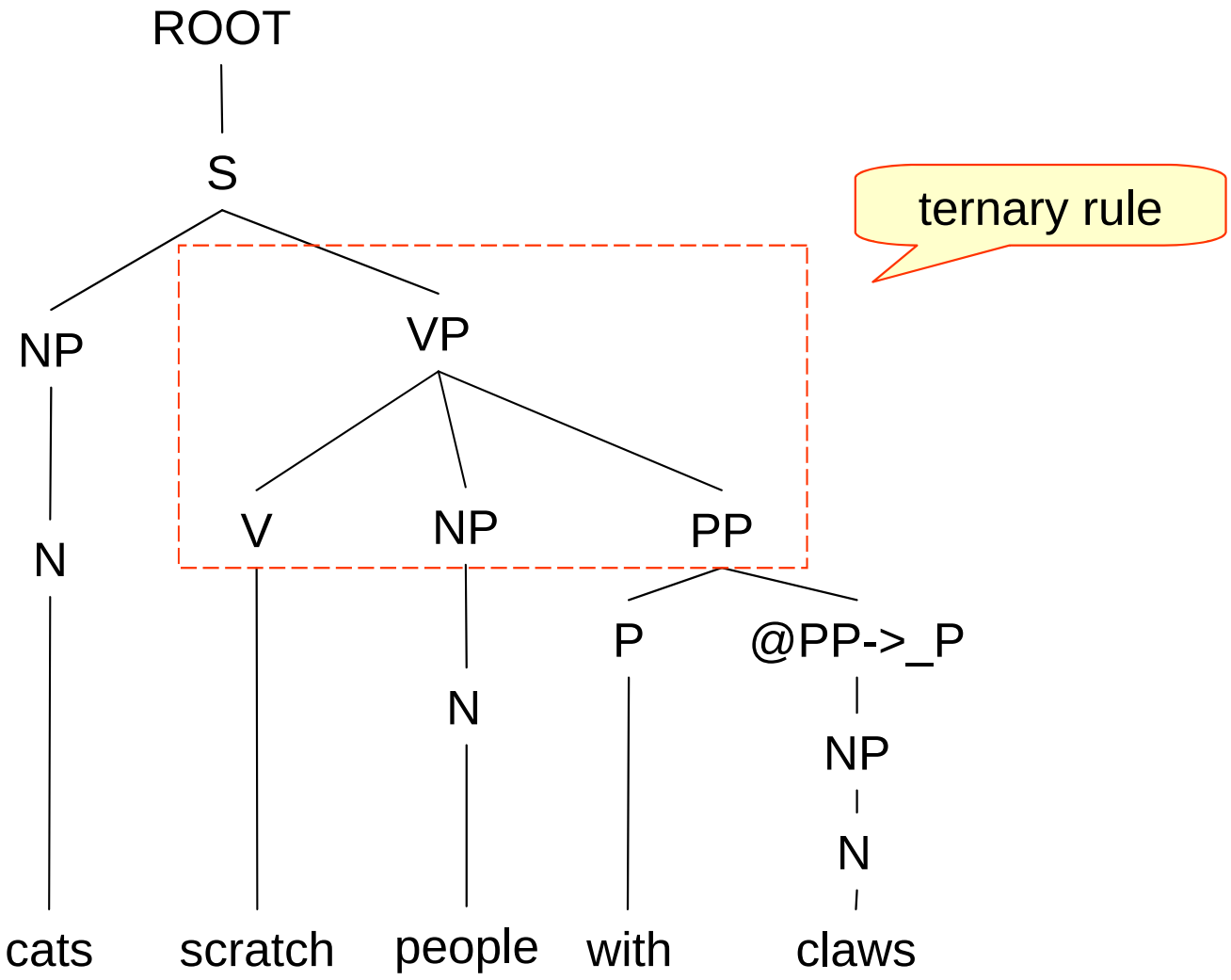
After binarization..

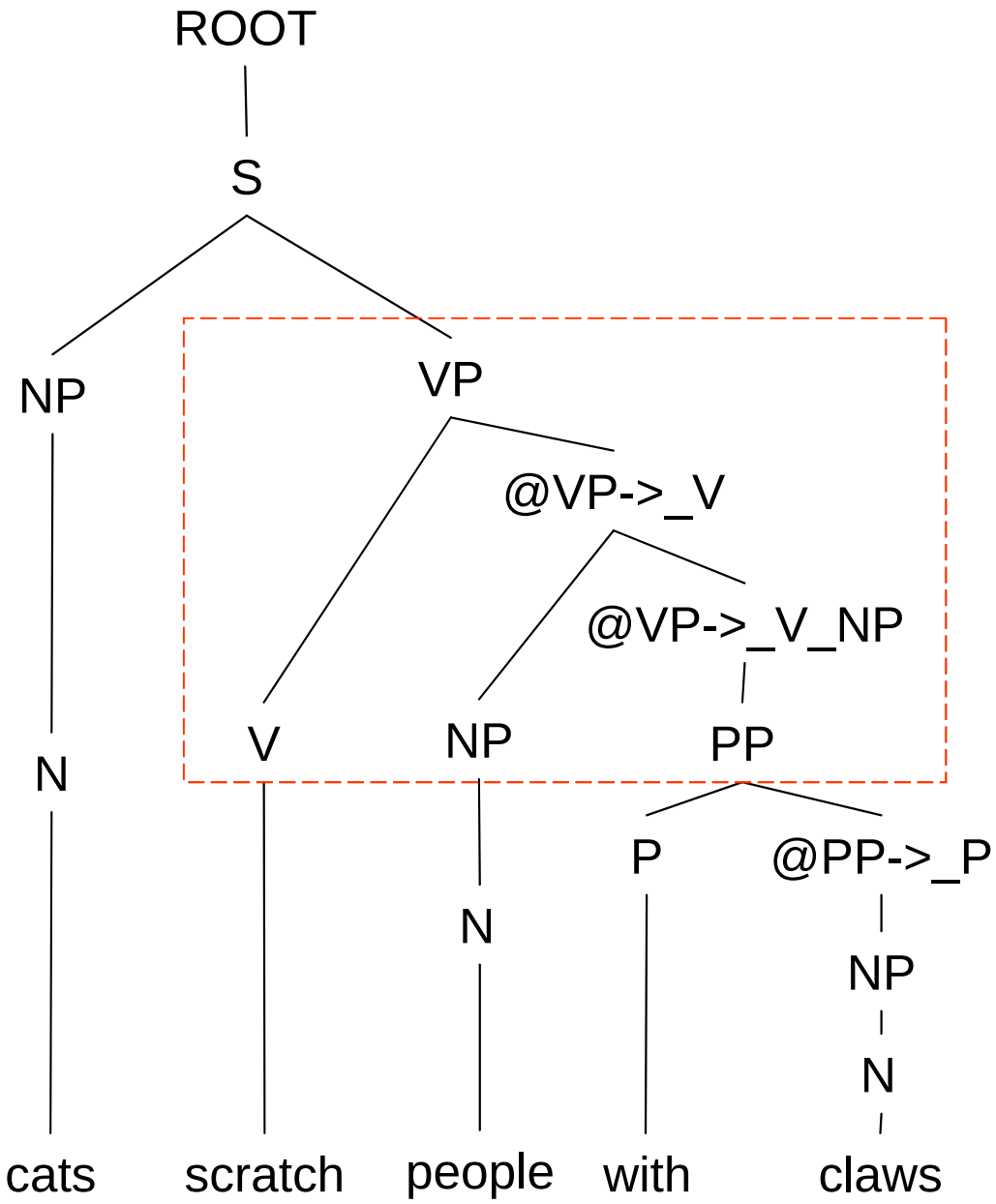


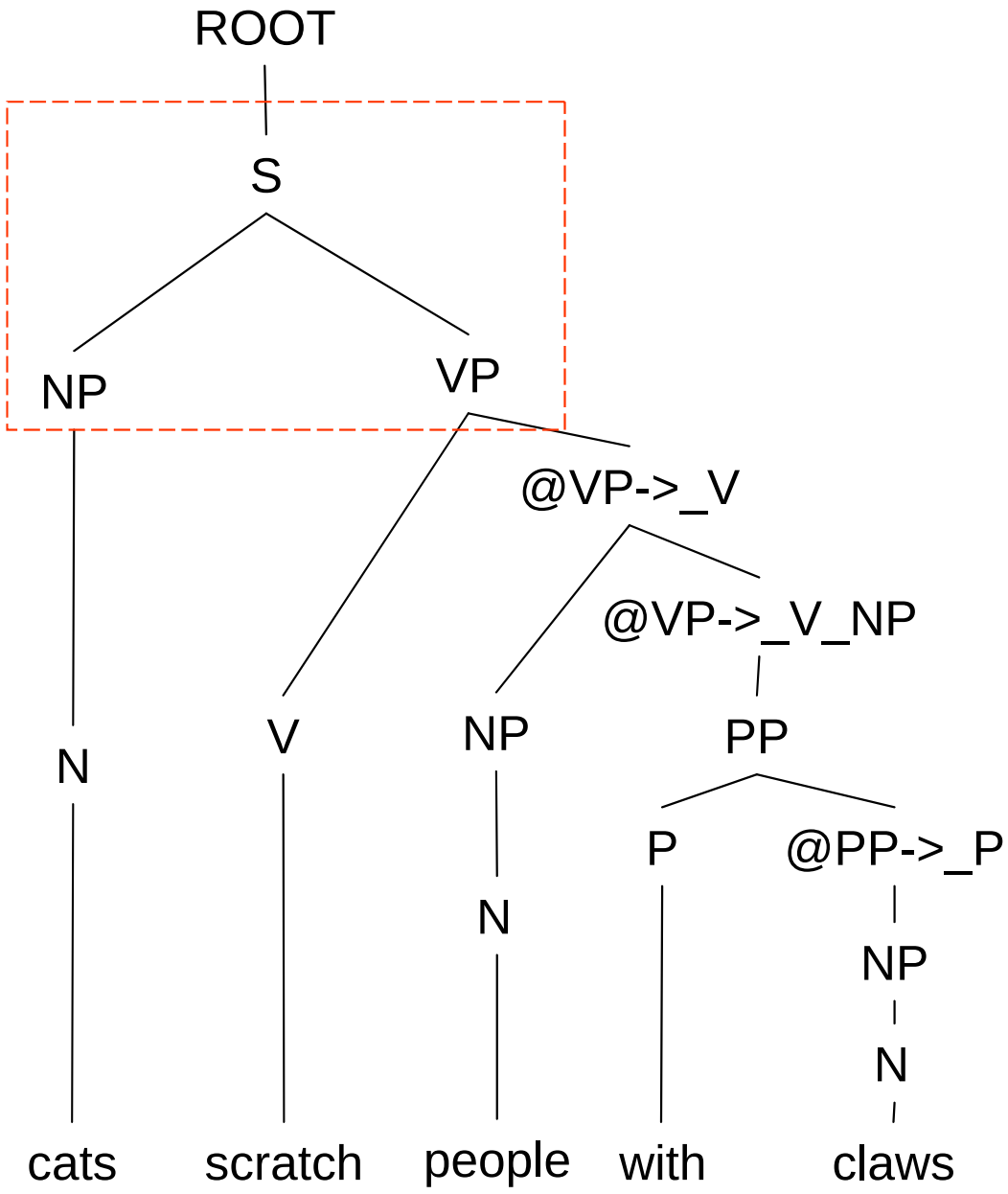

ROOT
S
NP
VP
N
V
NP
PP
P
NP
N
N
cats
scratch
people
with
claws
Binary rule

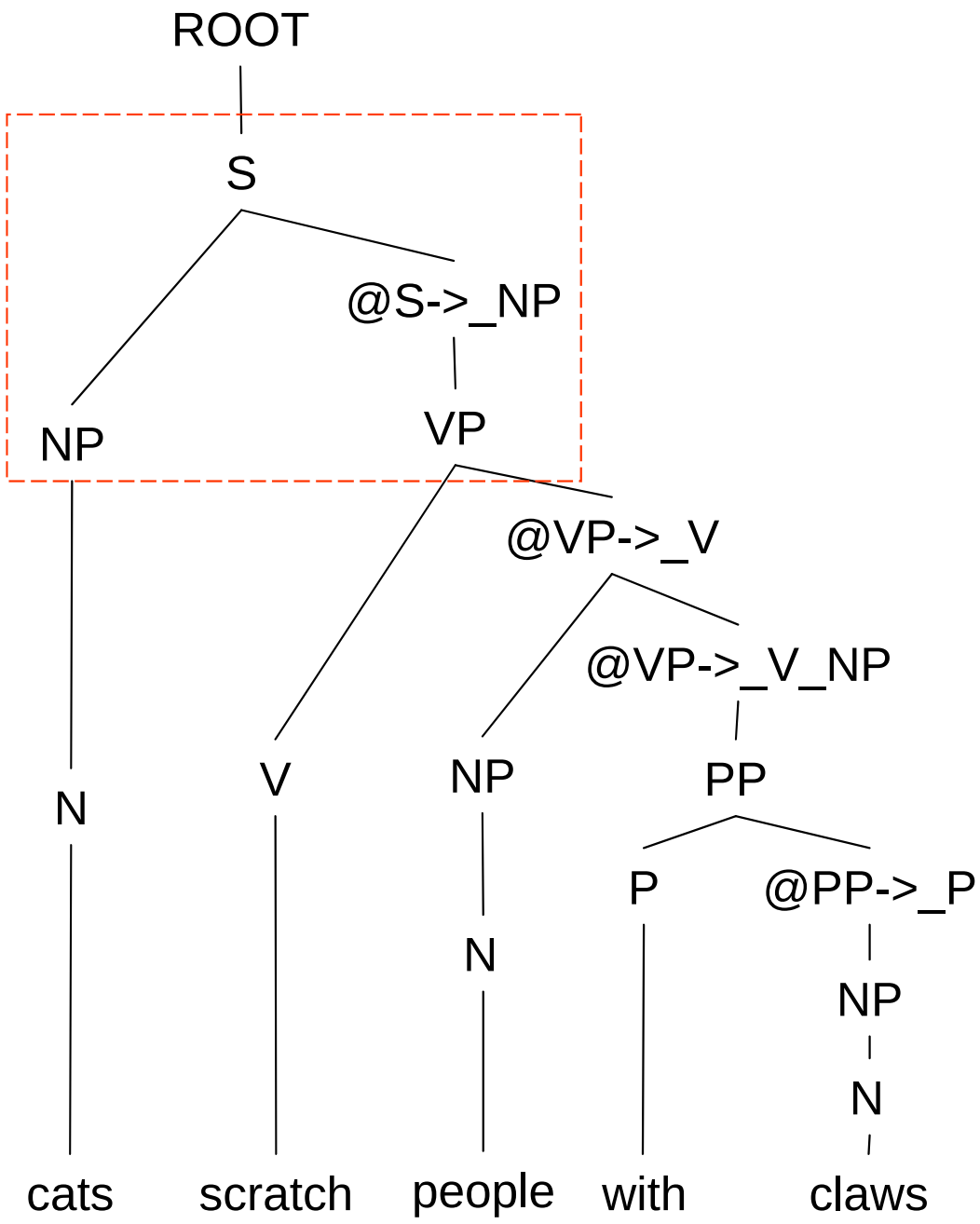


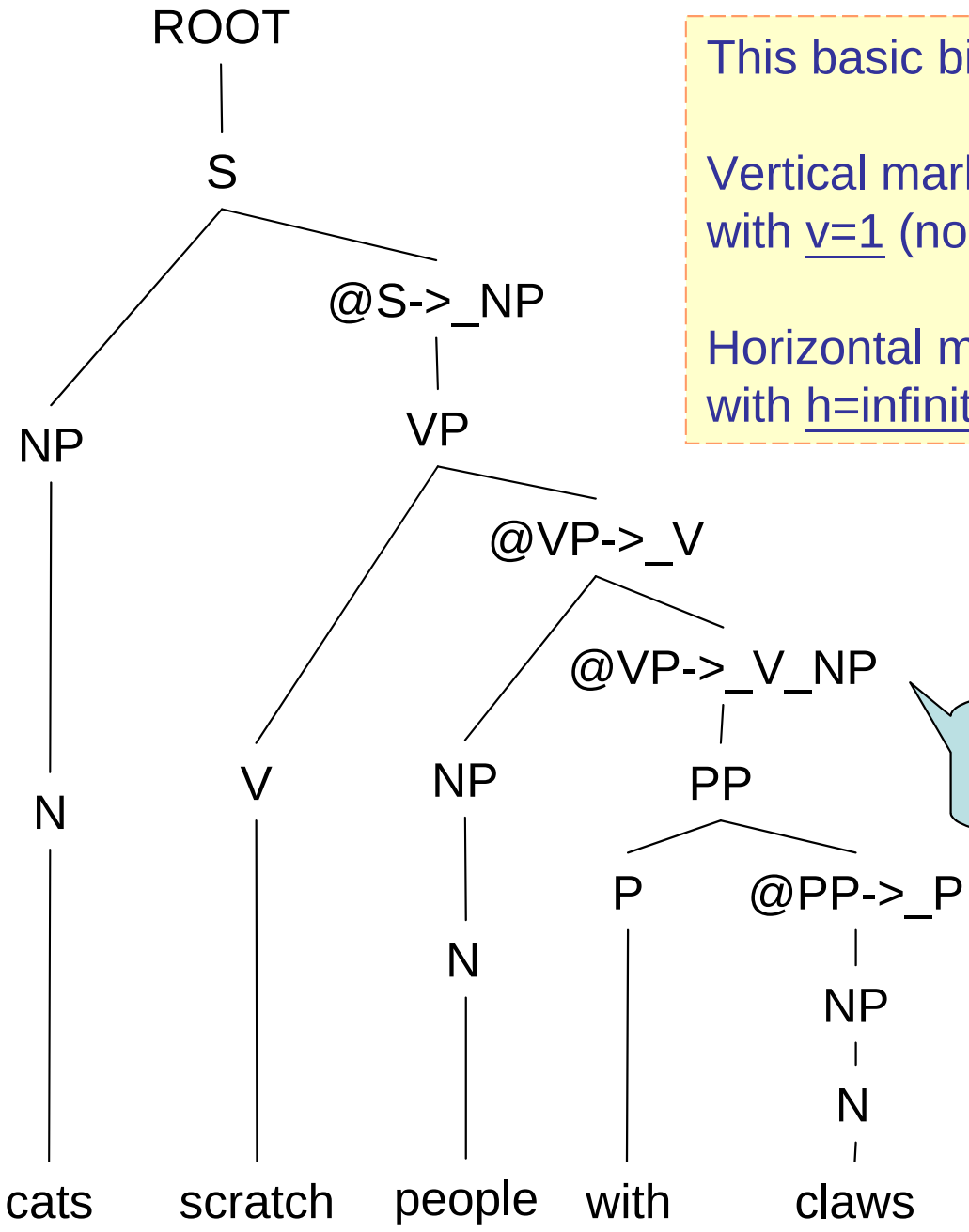
Seems redundant? (the rule was already binary)
Reason: easier to see how to make
finite-order horizontal markovizations
(explained later)


ROOT
S
NP
VP
N
V
NP
PP
P
@PP->_P
N
NP
N
cats
scratch
people
with
claws
ternary rule


ROOT
S
NP
VP
@VP->_V
@VP->_V_NP
N
V
NP
PP
P
@PP->_P
N
NP
N
cats
scratch
people
with
claws


ROOT
S
NP
VP
@VP->_V
@VP->_V_NP
N
V
NP
PP
P
@PP->_P
N
NP
N
cats
scratch
people
with
claws


ROOT
S
@S->_NP
NP
VP
@VP->_V
@VP->_V_NP
N
V
NP
PP
N
P
@PP->_P
NP
N
cats
scratch
people
with
claws



This basic binarization is equivalent to:

Vertical markovazation
with v=1 (no parent annotation)

Horizontal markovization
with h=infinity (not forgetting any siblings)

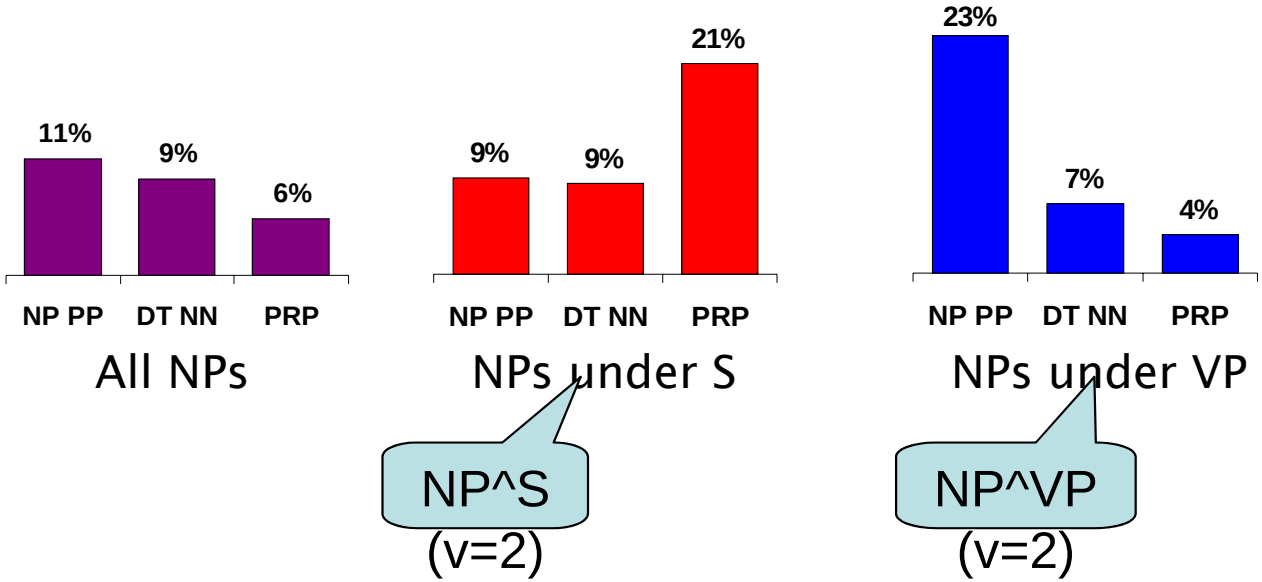
VP→V NP PP
Remembers 2 siblings

If there's a rule
VP → V NP PP PP

,
@VP->_V_NP_PP
will exist.

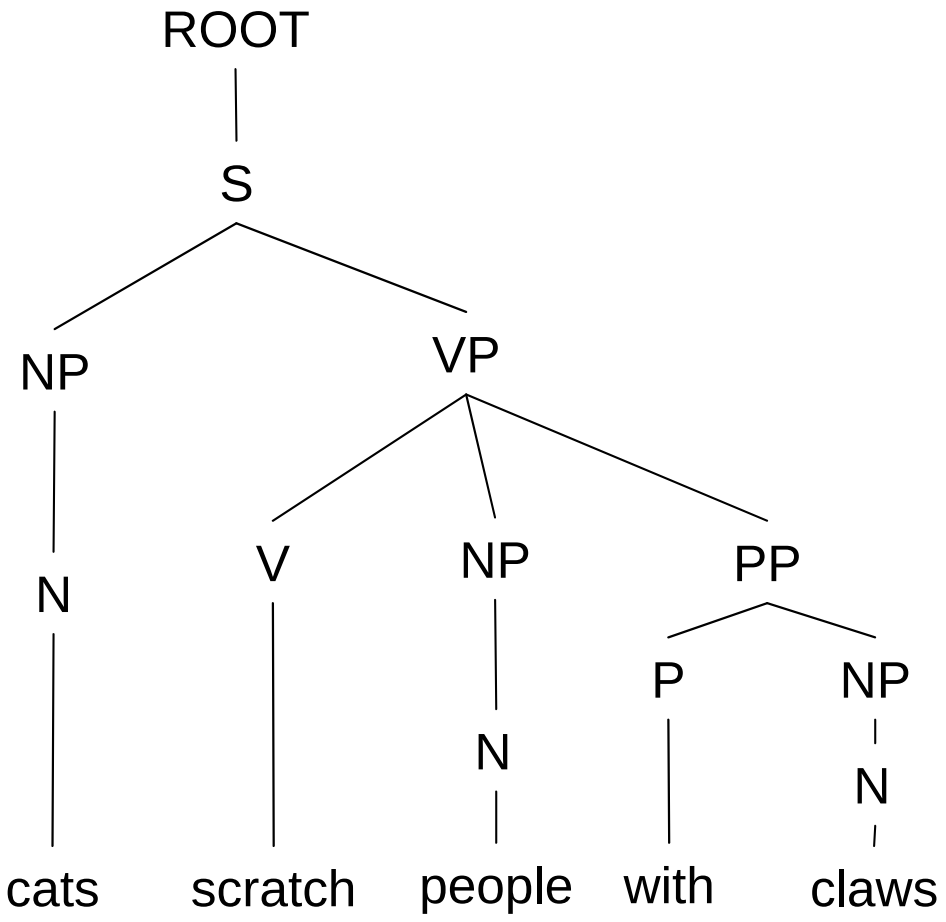
Two deficiencies of basic binarization (1/2)

1. PCFG independence assumption
   - Often too strong.
   - Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).

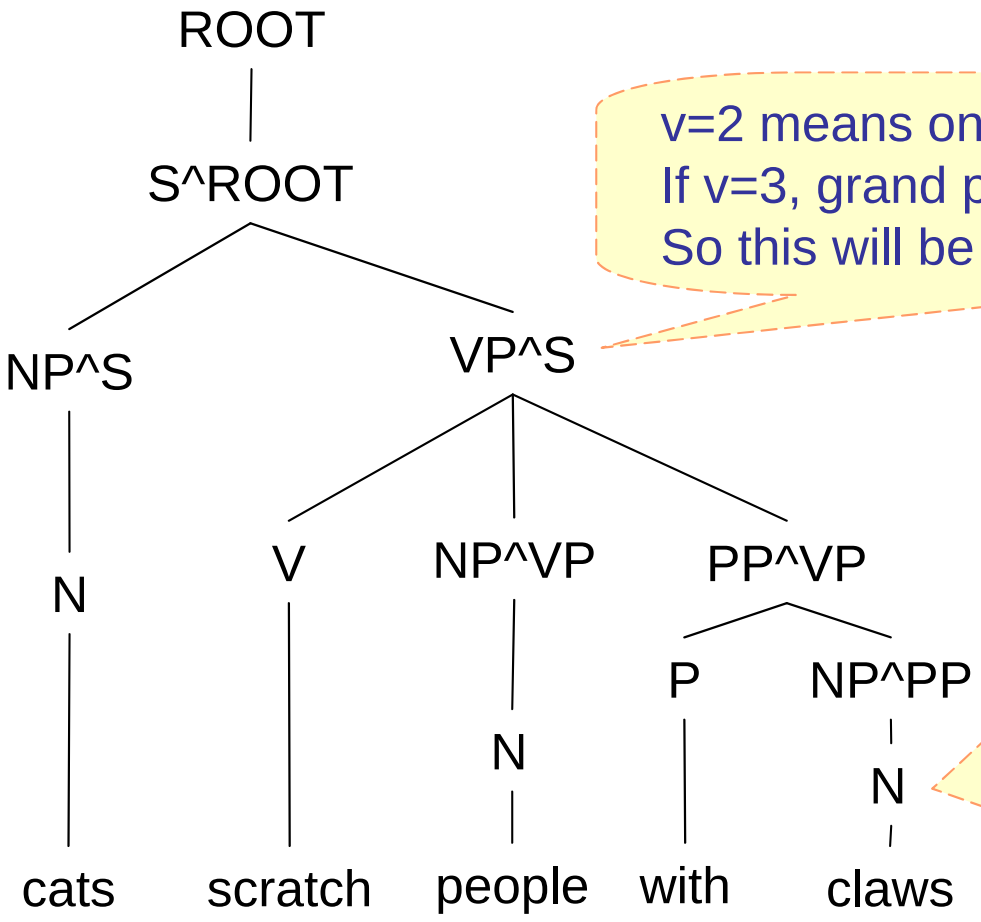


- Can be solved by “vertical markovization” (parent annotation)

Vertical Markovization (v=2): Before…



Vertical Markovization (v=2): After…

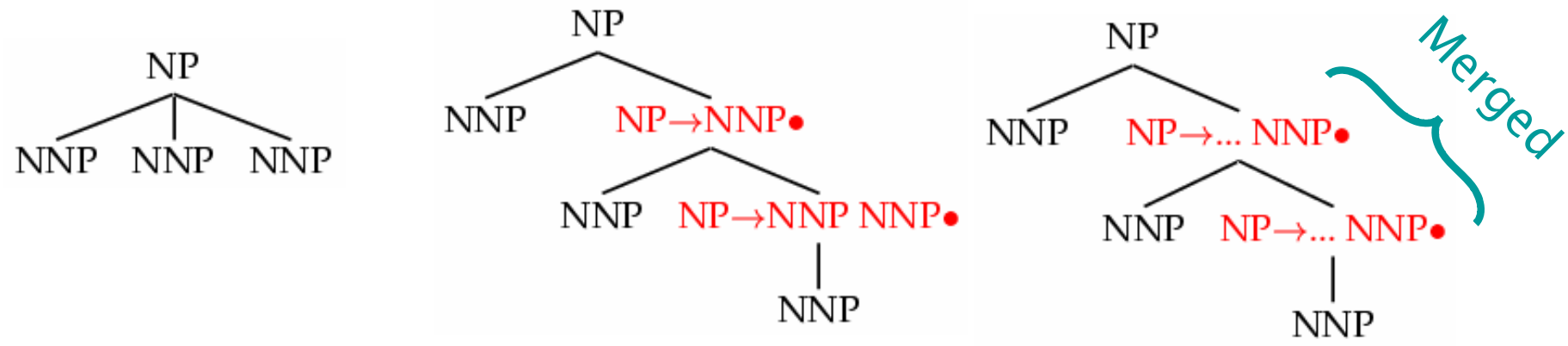


v=2 means only annotate with the parent node.
If v=3, grand parent node is also included.
So this will be **VP^S^ROOT**

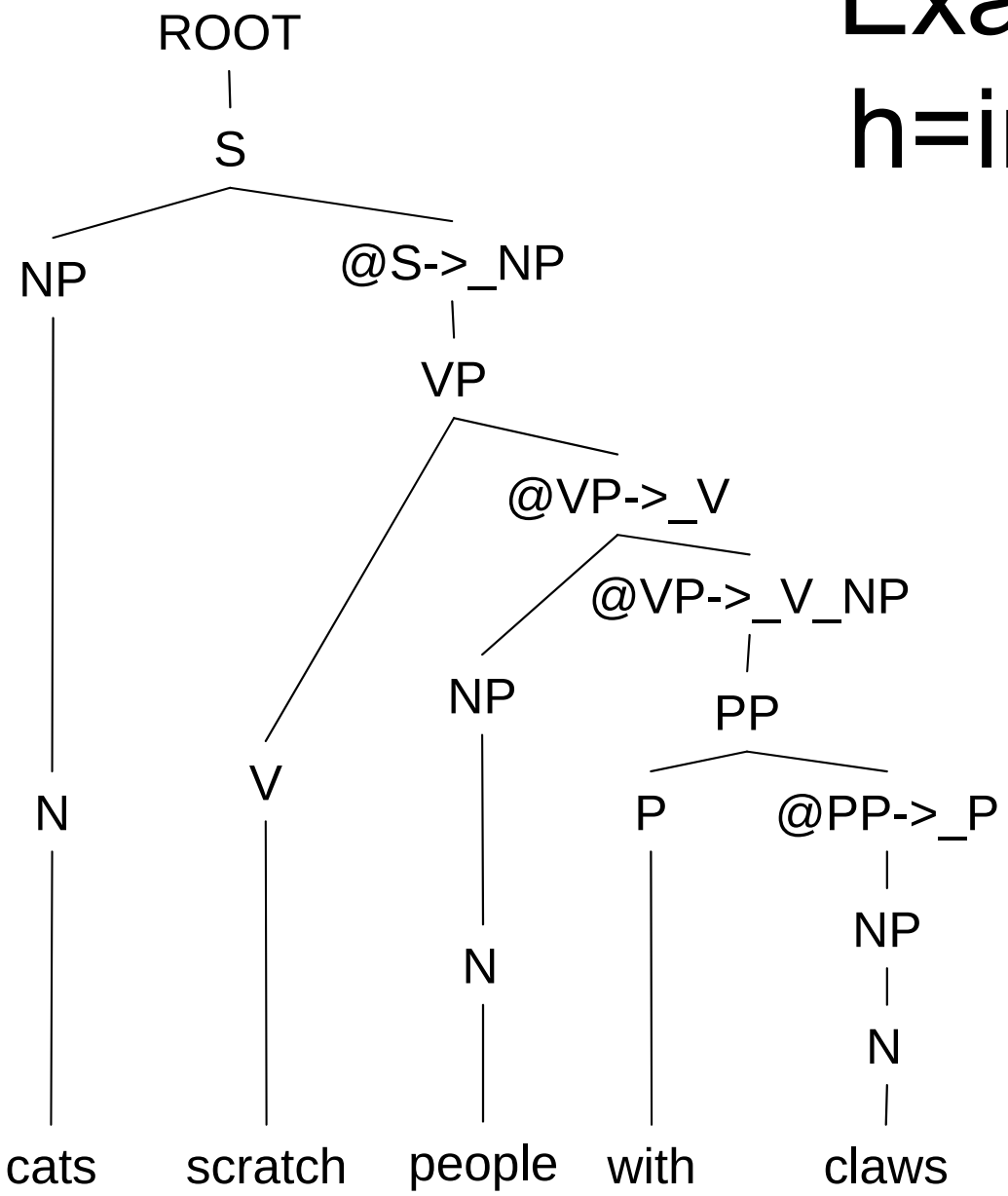
Suggestion:
Don't parent annotate POS tags.
You'll have to be more careful about smoothing the lexicon if you really want to.

Two deficiencies of basic binarization (2/2)

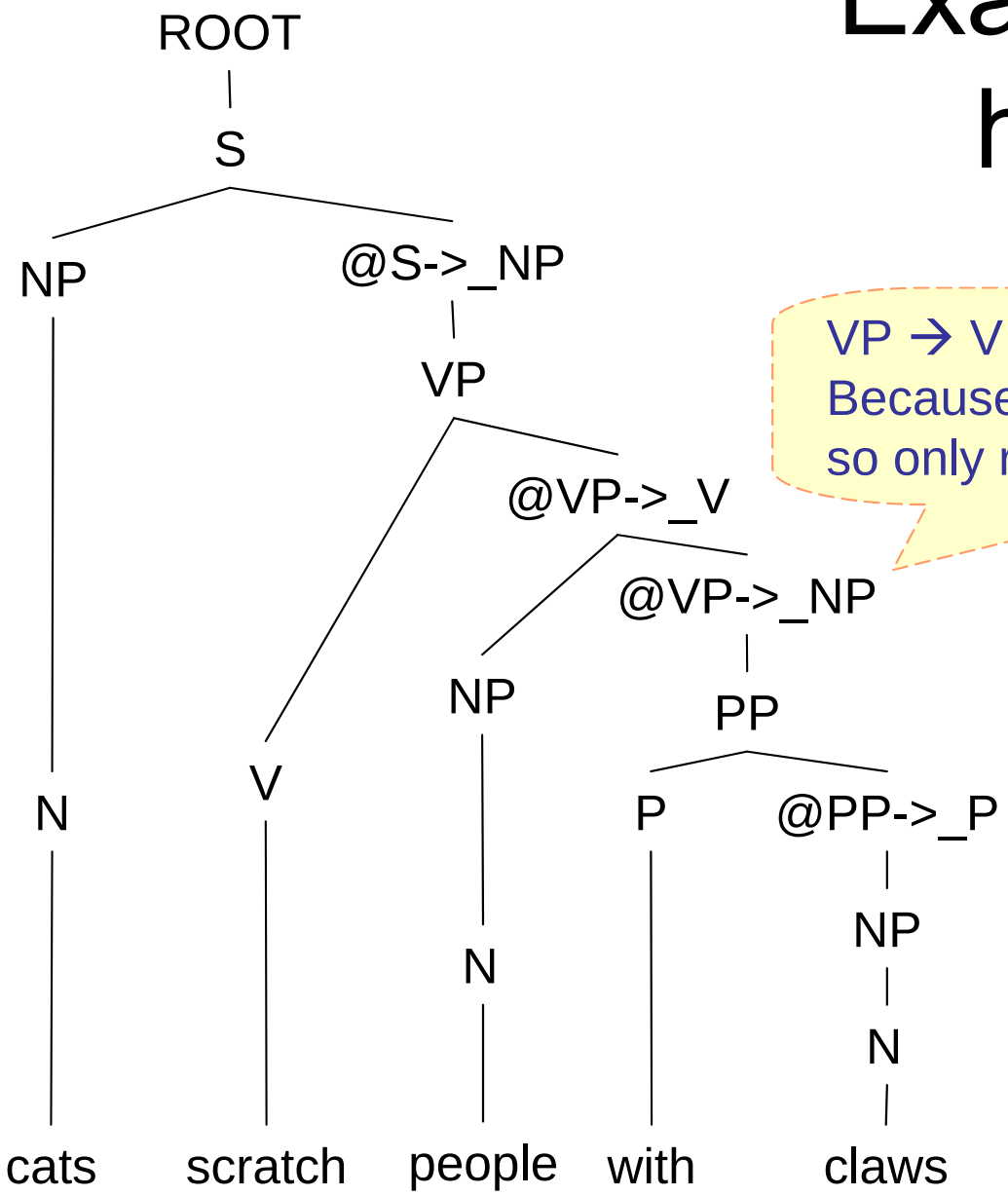
2. Many rules have been seen only once
   - Sparseness
   - We can make the horizontal markovization more forgetful.



Example: h=infinity



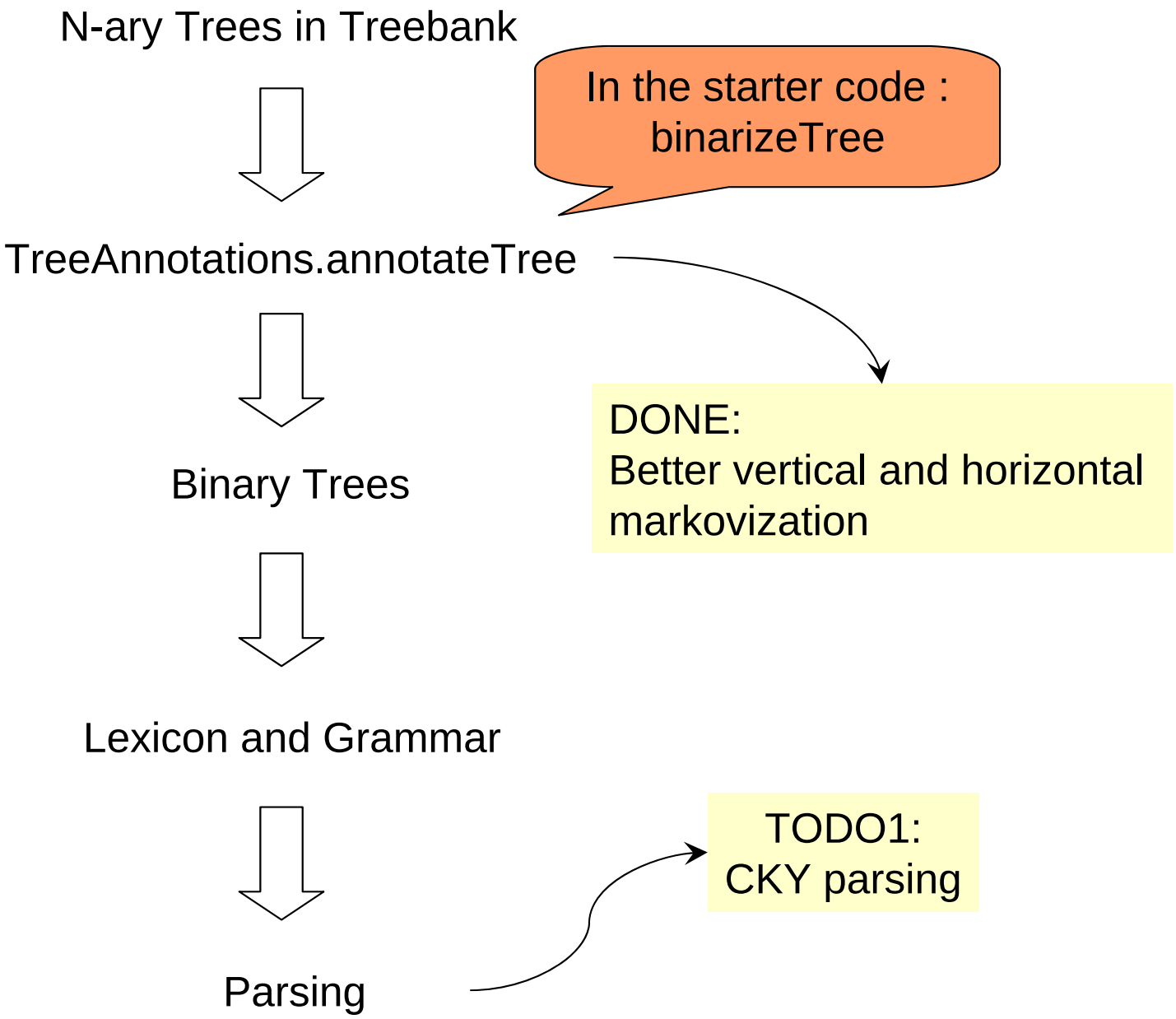
Example: h=1



VP → V NP PP
Because h=1,
so only remember “NP” but forget “V”

Some tips on Markovization

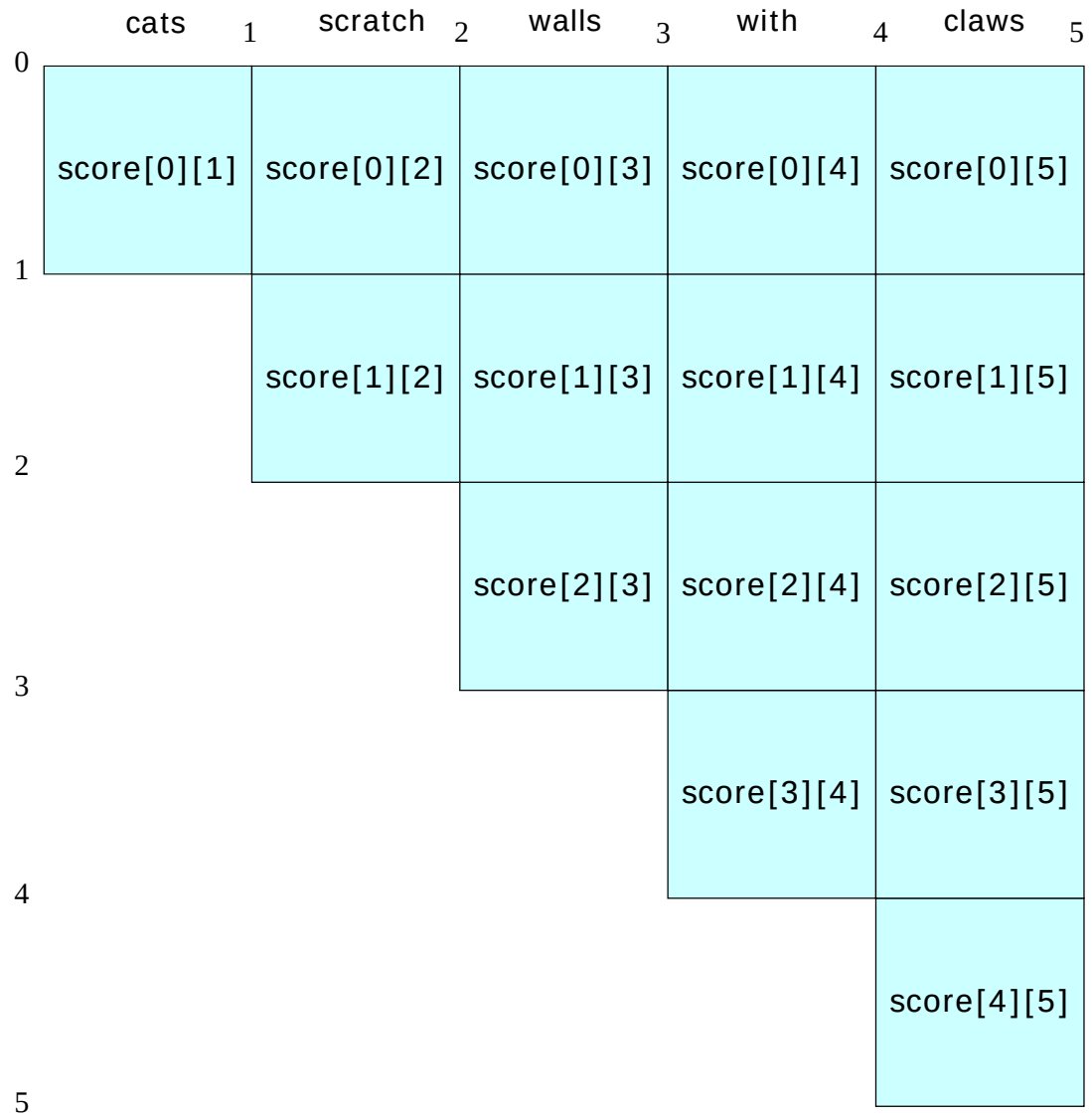
1. Vertical & horizontal
   - In the Stanford Parser, the order is: first do vertical markovization, and then horizontal markovization.
2. “unAnnotateTree” method
   - Although the comment said the unannotation cuts at the leftmost -, ^, or : character, but it actually cuts at ‘-’ or ‘=‘.
   - One solution: instead of “NP^S”, use “NP-^S” or “NP=S”
3. Don’t parent annotate POS tags.
   - Can be useful as well. But you need to do some fancier smoothing to get it to work well, and leaving it out will keep your grammar more compact.

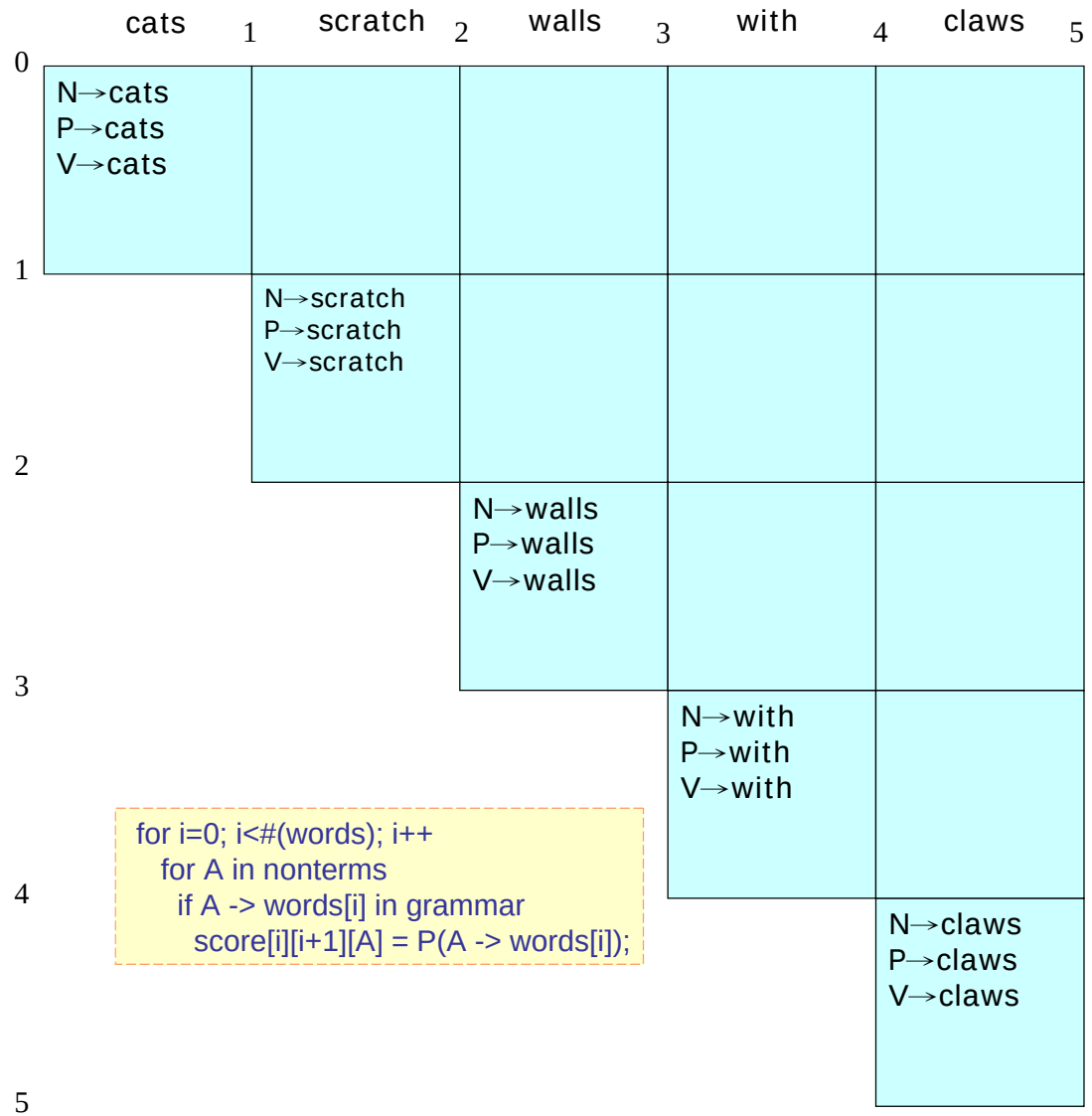

N-ary Trees in Treebank
In the starter code :
binarizeTree
TreeAnnotations.annotateTree
DONE:
Better vertical and horizontal
markovization
Binary Trees
Lexicon and Grammar
TODO1:
CKY parsing
Parsing

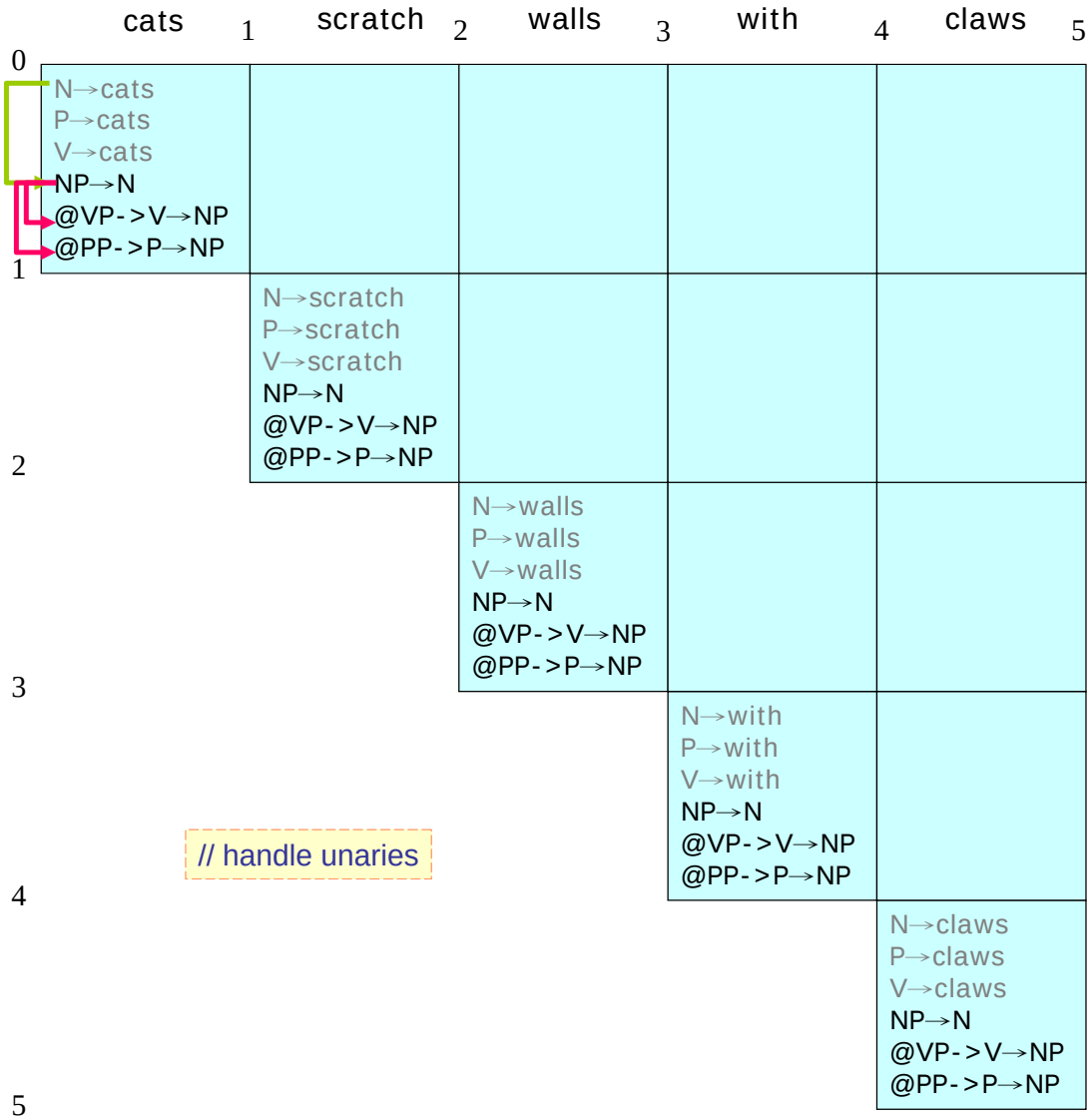
CKY algorithm

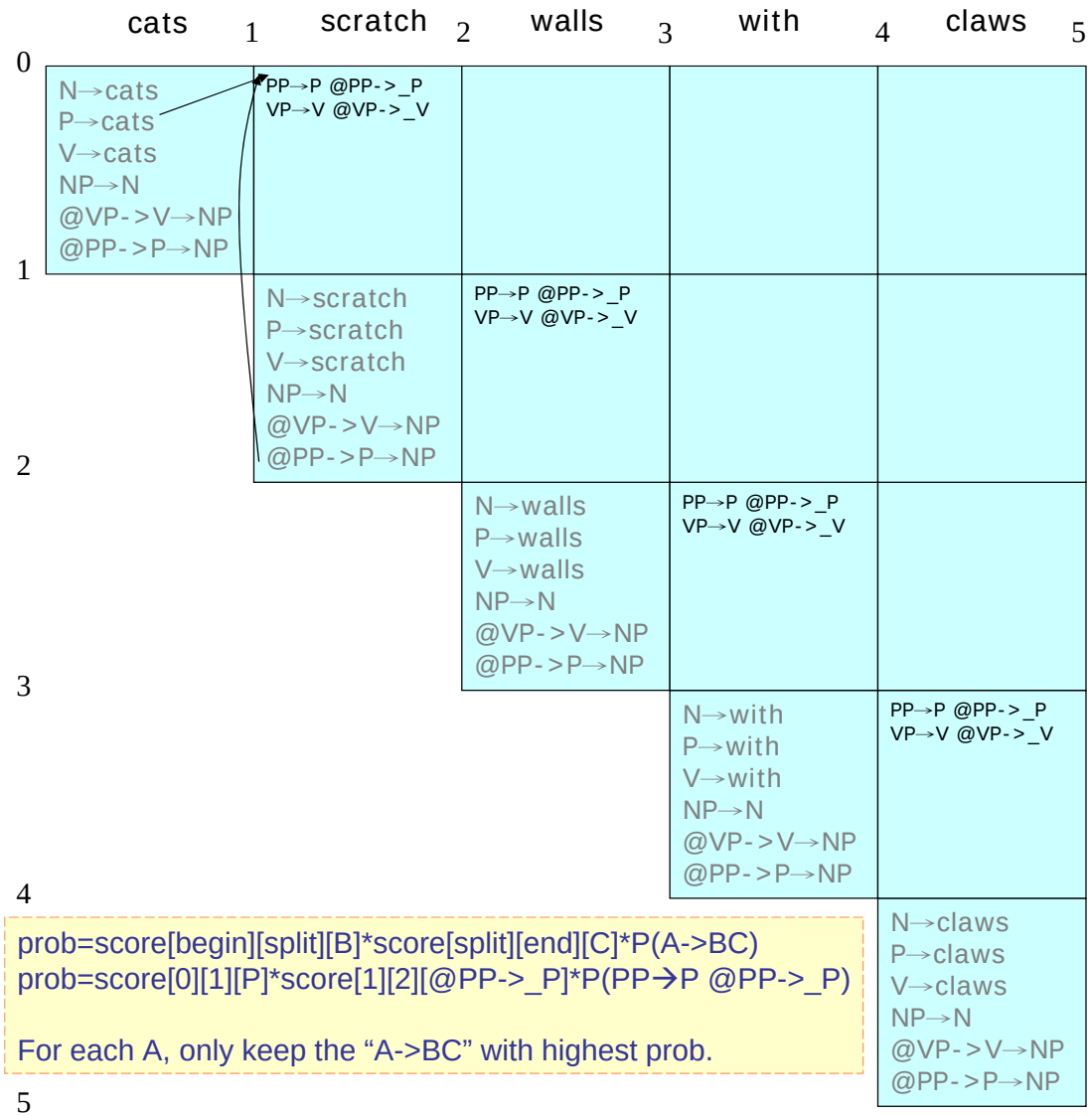
```
function CKY(words, grammar) returns most probable parse/probability
  score = new double[#(words)+1][#(words)+][#(nonterms)]
  back = new Pair[#(words)+1][#(words)+1][#nonterms]]
  for i=0; i<#(words); i++
    for A in nonterms
      if A -> words[i] in grammar
        score[i][i+1][A] = P(A -> words[i])
    //handle unaries
    boolean added = true
    while added
      added = false
      for A, B in nonterms
        if score[i][i+1][B] > 0 && A->B in grammar
          prob = P(A->B)*score[i][i+1][B]
          if(prob > score[i][i+1][A])
            score[i][i+1][A] = prob
            back[i][i+1] [A] = B
            added = true
```

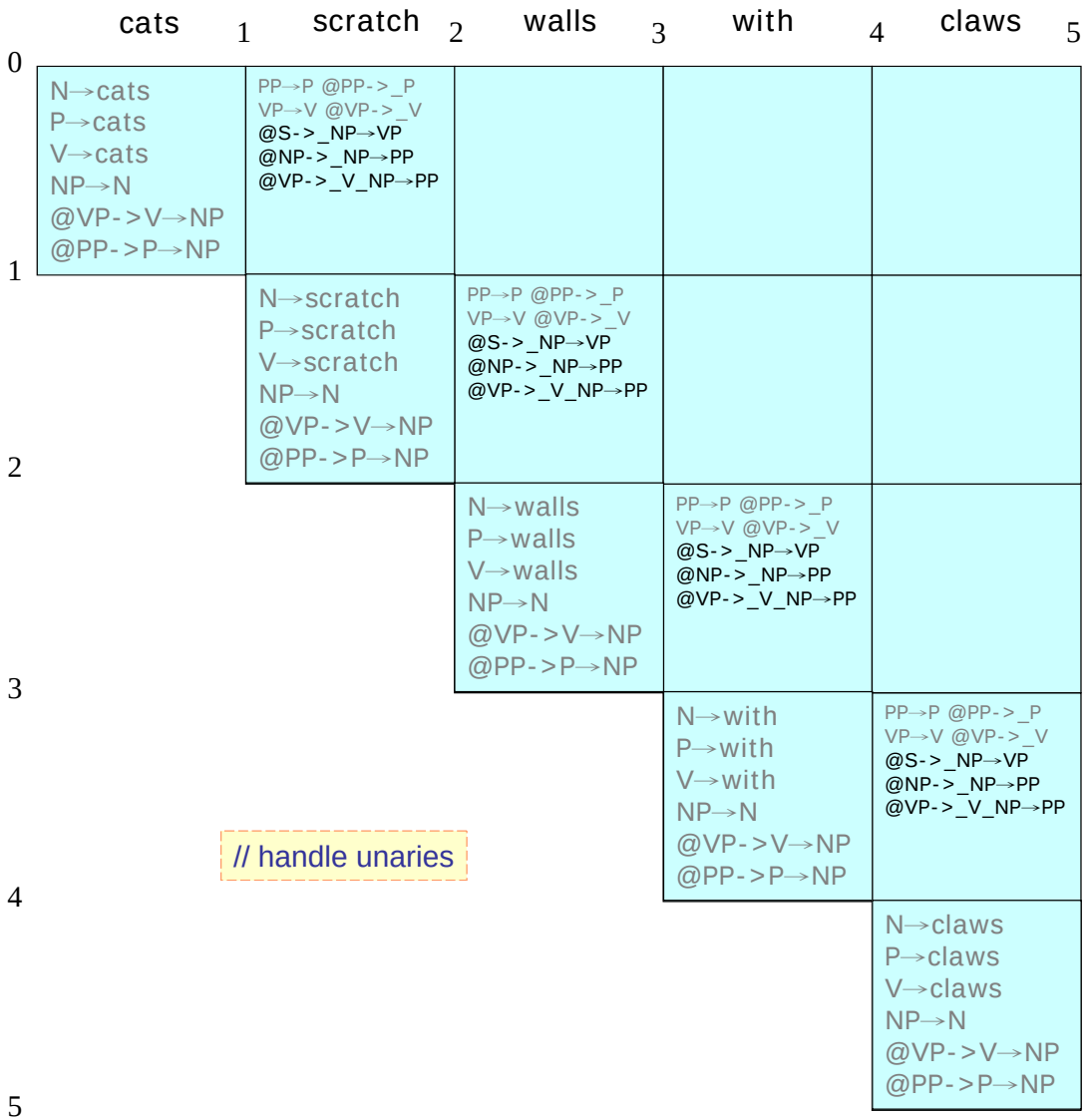
```
for span = 2 to #(words)
  for begin = 0 to #(words)- span
    end = begin + span
    for split = begin+1 to end-1
      for A,B,C in nonterms
        prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
        if(prob > score[begin][end][A])
          score[begin]end][A] = prob
          back[begin][end][A] = new Triple(split,B,C)
      //handle unaries
      boolean addes = true
      while added
        added = false
        for A, B in nonterms
          prob = P(A->B)*score[begin][end][B];
          if(prob > score[begin][end] [A])
            score[begin][end] [A] = prob
            back[begin][end] [A] = B
            added = true
return buildTree(score, back)
```


cats
1
scratch
2
walls
3
with
4
claws
5
0
1
2
3
4
5
score[0][1]
score[0][2]
score[0][3]
score[0][4]
score[0][5]
score[1][2]
score[1][3]
score[1][4]
score[1][5]
score[2][3]
score[2][4]
score[2][5]
score[3][4]
score[3][5]
score[4][5]

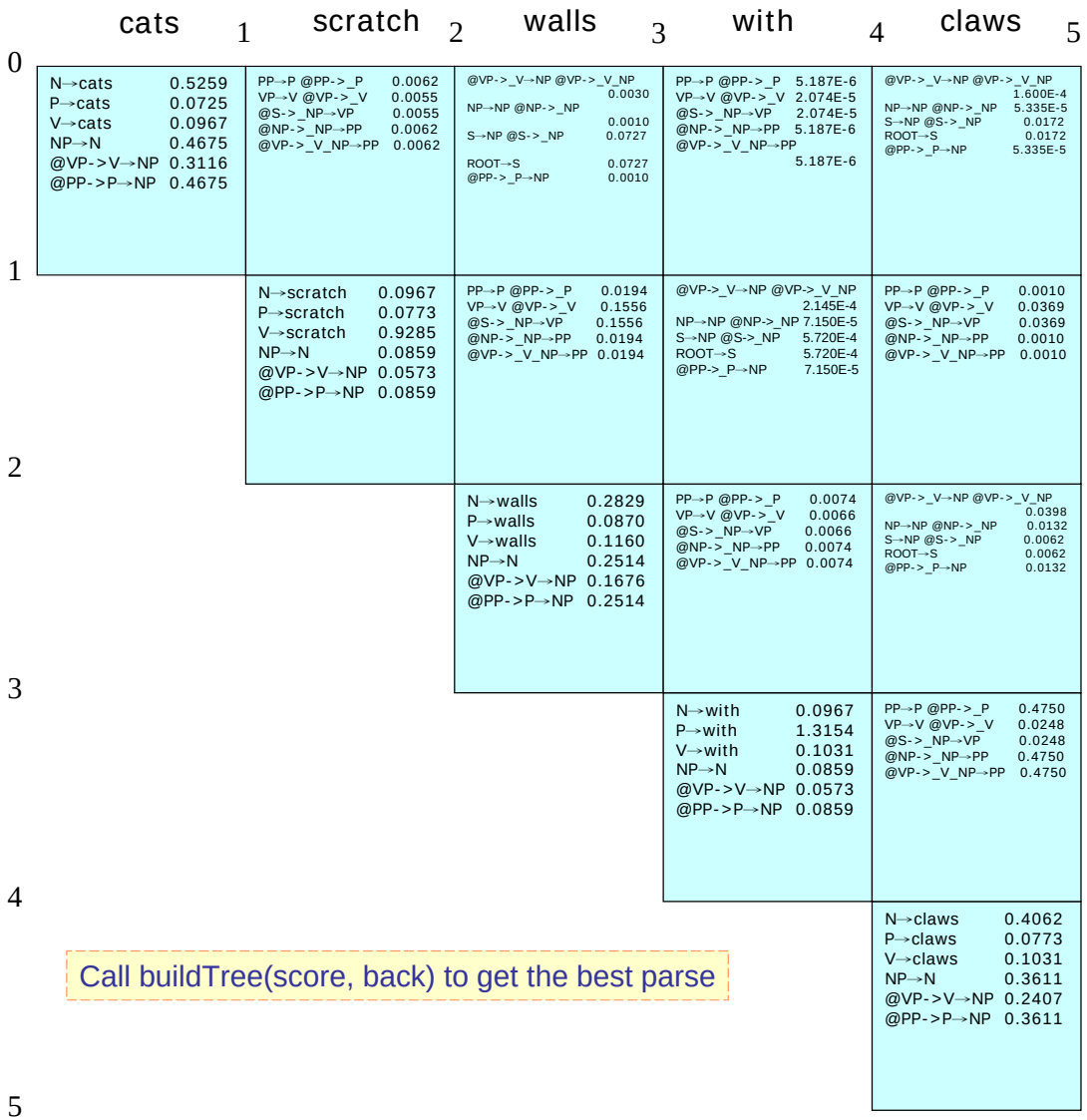

cats
1
scratch
2
walls
3
with
4
claws
5
0
1
2
3
4
5
N→cats
P→cats
V→cats
N→scratch
P→scratch
V→scratch
N→walls
P→walls
V→walls
N→with
P→with
V→with
N→claws
P→claws
V→claws
for i=0; i<#(words); i++
for A in nonterms
if A -> words[i] in grammar
score[i][i+1][A] = P(A -> words[i]);


cats
1
scratch
2
walls
3
with
4
claws
5
0
1
2
3
4
5
N→cats
P→cats
V→cats
NP→N
@VP->V→NP
@PP->P→NP
N→scratch
P→scratch
V→scratch
NP→N
@VP->V→NP
@PP->P→NP
N→walls
P→walls
V→walls
NP→N
@VP->V→NP
@PP->P→NP
N→with
P→with
V→with
NP→N
@VP->V→NP
@PP->P→NP
N→claws
P→claws
V→claws
NP→N
@VP->V→NP
@PP->P→NP
// handle unaries


cats
1
scratch
2
walls
3
with
4
claws
5
0
1
2
3
4
5
N→cats
P→cats
V→cats
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
N→scratch
P→scratch
V→scratch
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
N→walls
P→walls
V→walls
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
N→with
P→with
V→with
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
N→claws
P→claws
V→claws
NP→N
@VP->V→NP
@PP->P→NP
prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
prob=score[0][1][P]*score[1][2][@PP->_P]*P(PP→P @PP->_P)
For each A, only keep the “A->BC” with highest prob.


cats
1
scratch
2
walls
3
with
4
claws
5
0
1
2
3
4
5
N→cats
P→cats
V→cats
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
@S->_NP→VP
@NP->_NP→PP
@VP->_V_NP→PP
N→scratch
P→scratch
V→scratch
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
@S->_NP→VP
@NP->_NP→PP
@VP->_V_NP→PP
N→walls
P→walls
V→walls
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
@S->_NP→VP
@NP->_NP→PP
@VP->_V_NP→PP
N→with
P→with
V→with
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
@S->_NP→VP
@NP->_NP→PP
@VP->_V_NP→PP
N→claws
P→claws
V→claws
NP→N
@VP->V→NP
@PP->P→NP
// handle unaries

. . . . . .


cats
1
scratch
2
walls
3
with
4
claws
5
0
1
2
3
4
5
N→cats 0.5259
P→cats 0.0725
V→cats 0.0967
NP→N 0.4675
@VP->V→NP 0.3116
@PP->P→NP 0.4675
PP→P @PP->_P 0.0062
VP→V @VP->_V 0.0055
@S->_NP→VP 0.0055
@NP->_NP→PP 0.0062
@VP->_V_NP→PP 0.0062
@VP->_V→NP @VP->_V_NP 0.0030
NP→NP @NP->_NP 0.0010
S→NP @S->_NP 0.0727
ROOT→S 0.0727
@PP->_P→NP 0.0010
PP→P @PP->_P 5.187E-6
VP→V @VP->_V 2.074E-5
@S->_NP→VP 2.074E-5
@NP->_NP→PP 5.187E-6
@VP->_V_NP→PP 5.187E-6
@VP->_V→NP @VP->_V_NP 1.600E-4
NP→NP @NP->_NP 5.335E-5
S→NP @S->_NP 0.0172
ROOT→S 0.0172
@PP->_P→NP 5.335E-5
N→scratch 0.0967
P→scratch 0.0773
V→scratch 0.9285
NP→N 0.0859
@VP->V→NP 0.0573
@PP->P→NP 0.0859
PP→P @PP->_P 0.0194
VP→V @VP->_V 0.1556
@S->_NP→VP 0.1556
@NP->_NP→PP 0.0194
@VP->_V_NP→PP 0.0194
@VP->_V→NP @VP->_V_NP 2.145E-4
NP→NP @NP->_NP 7.150E-5
S→NP @S->_NP 5.720E-4
ROOT→S 5.720E-4
@PP->_P→NP 7.150E-5
PP→P @PP->_P 0.0010
VP→V @VP->_V 0.0369
@S->_NP→VP 0.0369
@NP->_NP→PP 0.0010
@VP->_V_NP→PP 0.0010
N→walls 0.2829
P→walls 0.0870
V→walls 0.1160
NP→N 0.2514
@VP->V→NP 0.1676
@PP->P→NP 0.2514
PP→P @PP->_P 0.0074
VP→V @VP->_V 0.0066
@S->_NP→VP 0.0066
@NP->_NP→PP 0.0074
@VP->_V_NP→PP 0.0074
@VP->_V→NP @VP->_V_NP 0.0398
NP→NP @NP->_NP 0.0132
S→NP @S->_NP 0.0062
ROOT→S 0.0062
@PP->_P→NP 0.0132
N→with 0.0967
P→with 1.3154
V→with 0.1031
NP→N 0.0859
@VP->V→NP 0.0573
@PP->P→NP 0.0859
PP→P @PP->_P 0.4750
VP→V @VP->_V 0.0248
@S->_NP→VP 0.0248
@NP->_NP→PP 0.4750
@VP->_V_NP→PP 0.4750
N→claws 0.4062
P→claws 0.0773
V→claws 0.1031
NP→N 0.3611
@VP->V→NP 0.2407
@PP->P→NP 0.3611
Call buildTree(score, back) to get the best parse