CFGs and PCFGs

(Probabilistic)
Context-Free
Grammars

A phrase structure grammar

Context Free Grammar (回想Compiler)

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → *e*

PP → P NP

N → people

N → fish

N → tanks

N → rods

V → people

V → fish

V → tanks

P → with

people fish tanks

people fish with rods

Phrase structure grammars = context-free grammars (CFGs)

- $G = (T, N, S, R)$
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol ($S \in N$)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup T)^*$

- A grammar G generates a language L.

Phrase structure grammars in NLP

T: 無法再拆成更小單位的符號，只會出現在RHS
C:
N: 可以再拆成更小單位的符號，出現在LHS或RHS
S: Start Symbol，代表句子的開頭
L: Phrase改寫成特定的字，例如DT -> the, NN -> man
R: Phrase改寫成Phrase，例如NP -> DT NN

- G = (T, C, N, S, L, R)
  - T is a set of terminal symbols
  - C is a set of preterminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol ($S \in N$)
  - L is the lexicon, a set of items of the form $X \rightarrow x$
    - $X \in P$ and $x \in T$
  - R is the grammar, a set of items of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup C)^*$
- By usual convention, S is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
- We usually write *e* for an empty sequence, rather than nothing

e代表Empty Symbol，代表句子的結束

A phrase structure grammar

Grammar

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → *e*

PP → P NP

Lexicon

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

people fish tanks

people fish with rods

Probabilistic – or stochastic – context-free grammars (PCFGs)

對於每一組Grammar Rule
找出機率和最高的組合
來代表這一句話的詞性結構

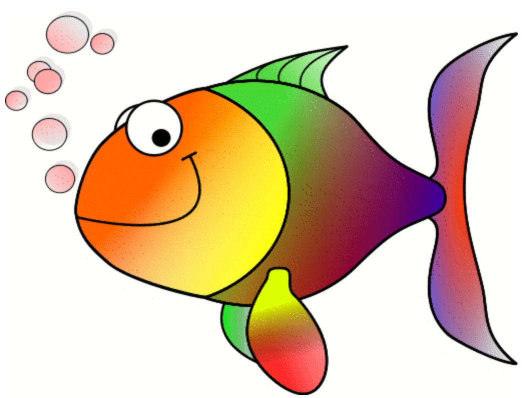
- G = (T, N, S, R, P)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol ($S \in N$)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
  - P is a probability function
    - P: $R \rightarrow [0,1]$
    - $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$

- A grammar G generates a language model L.

$$\sum_{\gamma \in T*} P(\gamma) = 1$$

A PCFG

$S \rightarrow NP\ VP$	1.0
$VP \rightarrow V\ NP$	0.6
$VP \rightarrow V\ NP\ PP$	0.4
$NP \rightarrow NP\ NP$	0.1
$NP \rightarrow NP\ PP$	0.2
$NP \rightarrow N$	0.7
$PP \rightarrow P\ NP$	1.0



N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

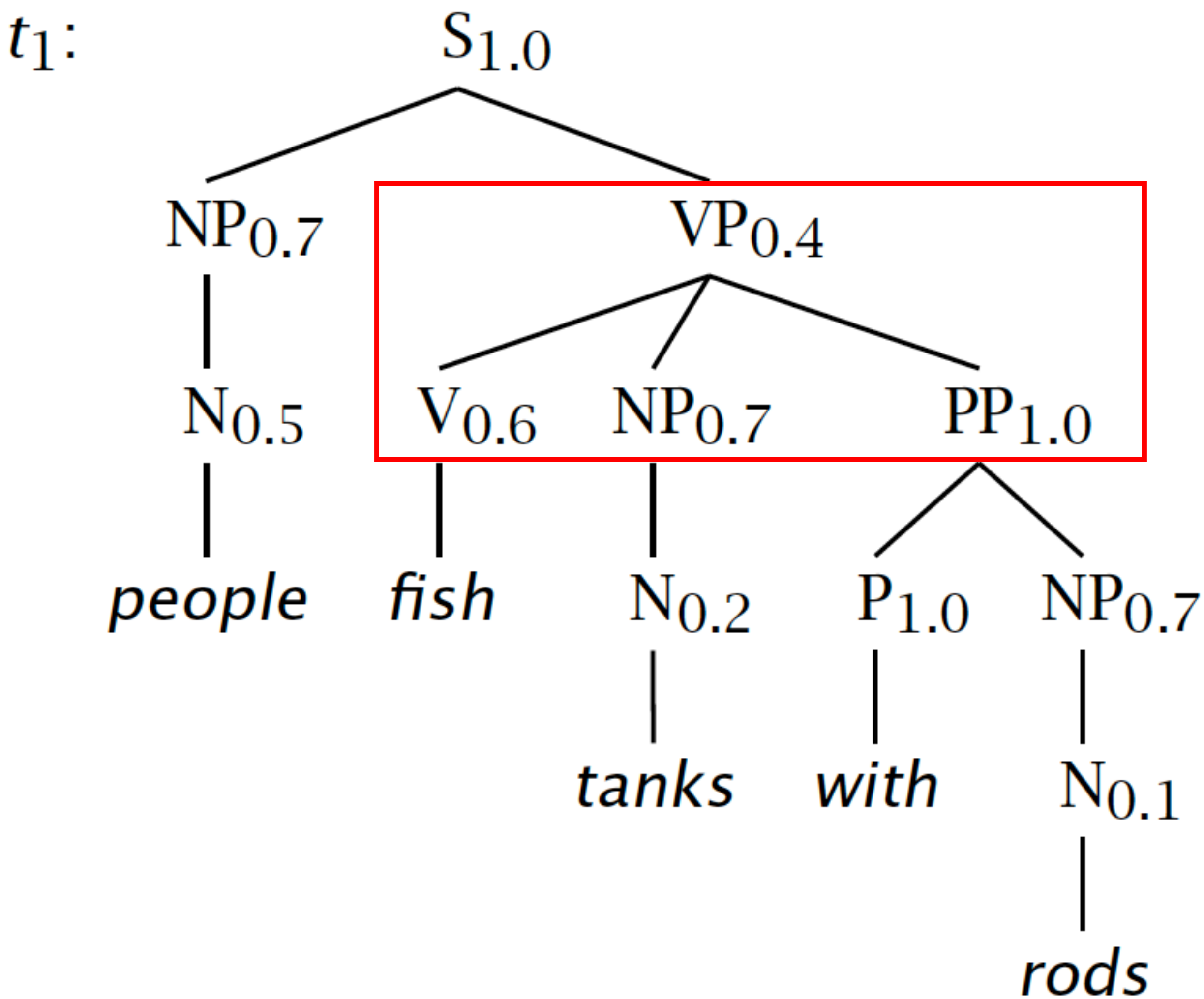
[With empty NP removed so less ambiguous]

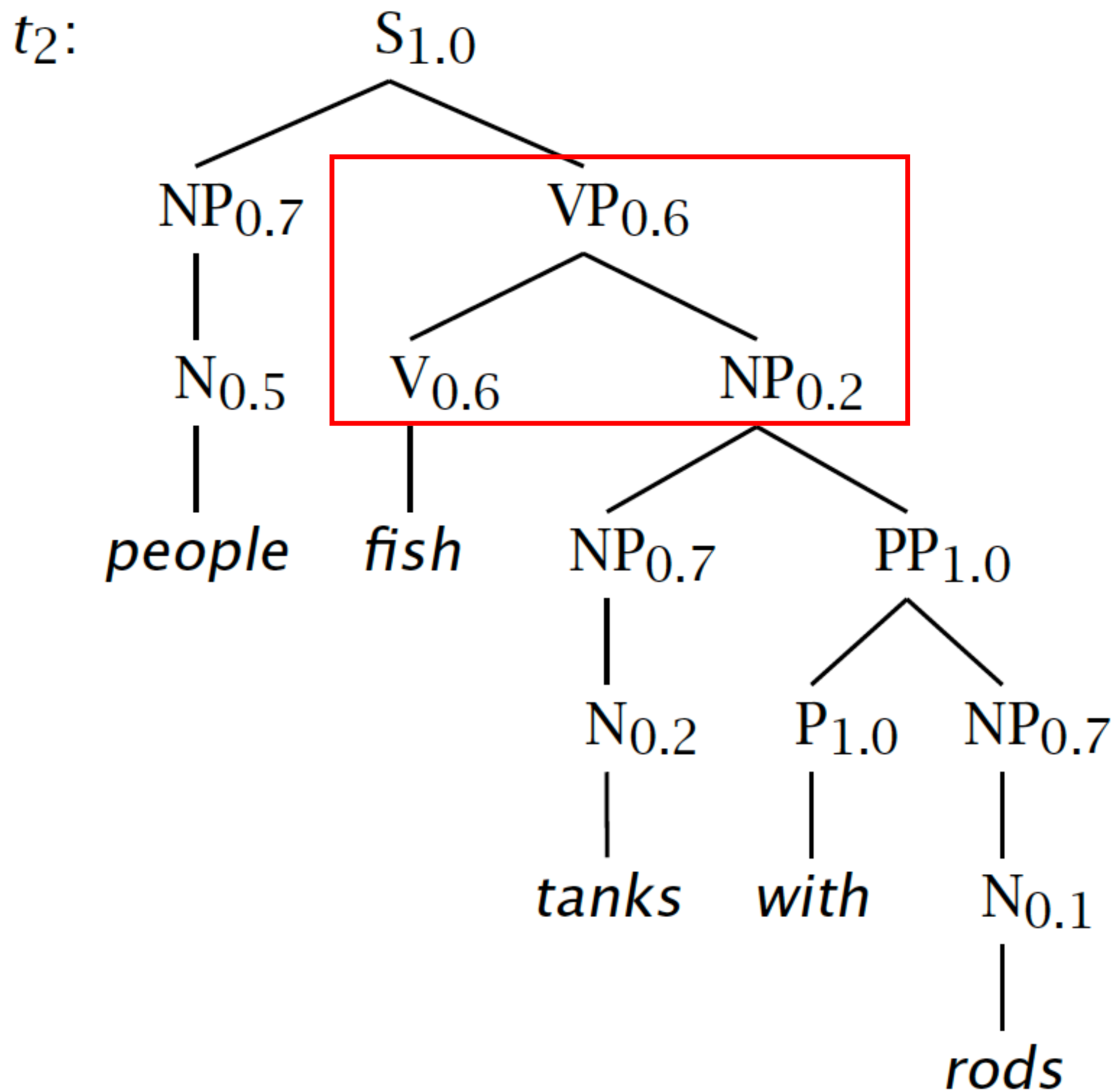
The probability of trees and strings

- P(t) – The probability of a tree t is the product of the probabilities of the rules used to generate it.
- P(s) – The probability of the string s is the sum of the probabilities of the trees which have that string as their yield

$$P(s) = \Sigma_j P(s, t) \text{ where } t \text{ is a parse of } s$$
$$= \Sigma_j P(t)$$

P(t)代表這一句話的詞性結構所畫出的樹為t的機率，用前一頁的機率去算
P(s)代表這一句話具有正確文法結構的機率（？）


t_1:
$S_{1.0}$
$NP_{0.7}$
$VP_{0.4}$
$N_{0.5}$
$V_{0.6}$
$NP_{0.7}$
$PP_{1.0}$
people
fish
$N_{0.2}$
$P_{1.0}$
$NP_{0.7}$
tanks
with
$N_{0.1}$
rods


t2:
S1.0
NP0.7
VP0.6
N0.5
V0.6
NP0.2
people
fish
NP0.7
PP1.0
N0.2
P1.0
NP0.7
tanks
with
N0.1
rods

Christopher Manning

Tree and String Probabilities

- *s = people fish tanks with rods*
- $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7$
  $\times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  $= 0.0008232$

Verb attach

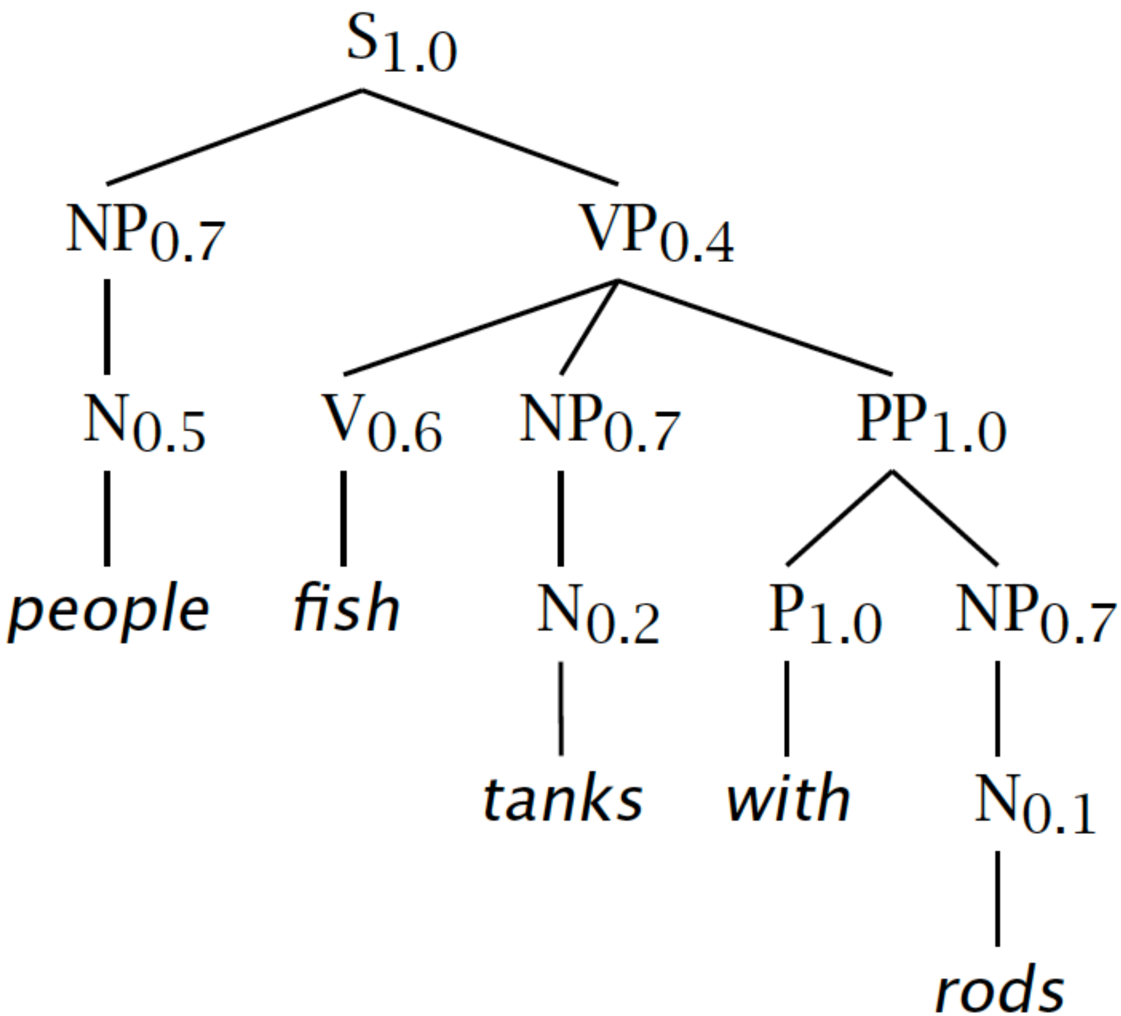
- $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2$
  $\times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  $= 0.00024696$

Noun attach

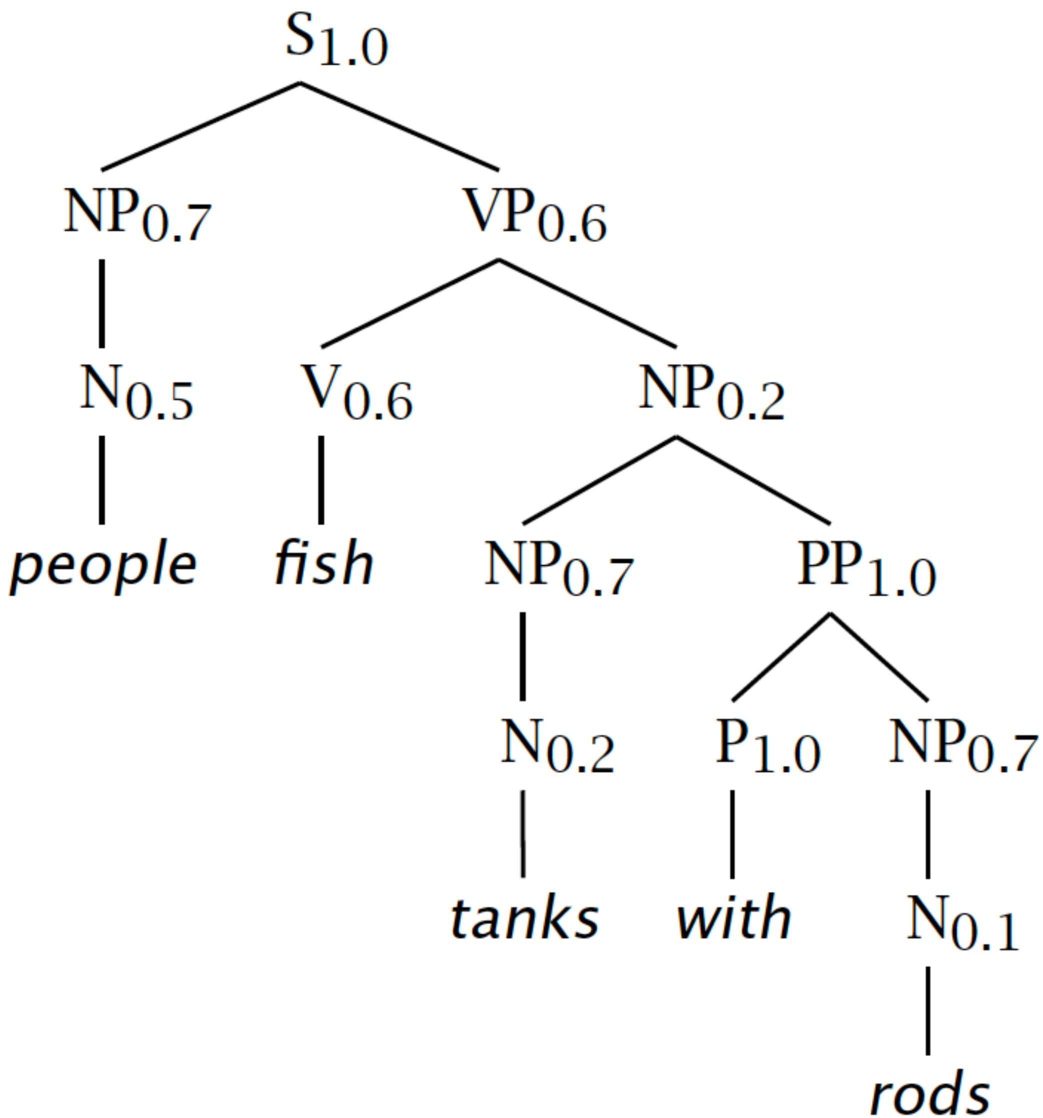
- $P(s) = P(t_1) + P(t_2)$
  $= 0.0008232 + 0.00024696$
  $= 0.00107016$

P(t1) > P(t2)
所以t1會是比較可能的詞性排列組合

t_1:


S1.0
NP0.7
VP0.4
N0.5
V0.6
NP0.7
PP1.0
people
fish
N0.2
P1.0
NP0.7
tanks
with
N0.1
rods

t_2:


S1.0
NP0.7
VP0.6
N0.5
V0.6
NP0.2
people
fish
NP0.7
PP1.0
N0.2
P1.0
NP0.7
tanks
with
N0.1
rods

CFGs and PCFGs

(Probabilistic) Context-Free Grammars

Grammar Transforms

Restricting the grammar form for efficient parsing

Chomsky Normal Form

試著把大量的Rules做合併
做出數量較少的Rules

- All rules are of the form $X \rightarrow Y\ Z$ or $X \rightarrow w$
  - $X, Y, Z \in N$ and $w \in T$
- A transformation to this form doesn't change the weak generative capacity of a CFG
  - That is, it recognizes the same language
    - But maybe with different trees
- Empties and unaries are removed recursively
- n-ary rules are divided by introducing new nonterminals (n > 2)

修改規則：
1. RHS只有e的要被拿掉
2. RHS只有一項，且可以被改寫為terminal的，要改寫
3. RHS有三項或以上的，要改成兩項
結果：所有Rules的RHS都應該只有兩項，或是只有一項terminal

A phrase structure grammar

S -> NP VP

S -> VP

S → NP VP	N → *people*
VP → V NP	N → *fish*
VP → V NP PP	N → *tanks*
NP → NP NP	N → *rods*
NP → NP PP	V → *people*
NP → N	V → *fish*
~~NP → *e*~~	V → *tanks*
PP → P NP	P → *with*

Chomsky Normal Form steps

$S \rightarrow NP\ VP$

$\cancel{S \rightarrow VP}$

$VP \rightarrow V\ NP$

$VP \rightarrow V$

$VP \rightarrow V\ NP\ PP$

$VP \rightarrow V\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow PP$

$NP \rightarrow N$

$PP \rightarrow P\ NP$

$PP \rightarrow P$

```
S -> V NP
S -> V
S -> V NP PP
S -> V PP
```

$N \rightarrow$ *people*

$N \rightarrow$ *fish*

$N \rightarrow$ *tanks*

$N \rightarrow$ *rods*

$V \rightarrow$ *people*

$V \rightarrow$ *fish*

$V \rightarrow$ *tanks*

$P \rightarrow$ *with*

Chomsky Normal Form steps

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$S \rightarrow V\ NP$

$VP \rightarrow V$

$\cancel{S \rightarrow V}$

$VP \rightarrow V\ NP\ PP$

$S \rightarrow V\ NP\ PP$

$VP \rightarrow V\ PP$

$S \rightarrow V\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow PP$

$NP \rightarrow N$

$PP \rightarrow P\ NP$

$PP \rightarrow P$

$N \rightarrow$ *people*

$N \rightarrow$ *fish*

$N \rightarrow$ *tanks*

$N \rightarrow$ *rods*

$V \rightarrow$ *people*

$V \rightarrow$ *fish*

$V \rightarrow$ *tanks*

$P \rightarrow$ *with*

```
S -> people
S -> fish
S -> tanks
```

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
~~VP → V~~
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
V → *fish*
S → *fish*
V → *tanks*
S → *tanks*
P → *with*

```
VP -> people
VP -> fish
VP -> tanks
```

Chomsky Normal Form steps

S → NP VP

VP → V NP

S → V NP

VP → V NP PP

S → V NP PP

VP → V PP

S → V PP

NP → NP NP

~~NP → NP~~

NP → NP PP

NP → PP

~~NP → N~~

PP → P NP

PP → P

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

S → *people*

VP → *people*

V → *fish*

S → *fish*

VP → *fish*

V → *tanks*

S → *tanks*

VP → *tanks*

P → *with*

```
NP -> people
NP -> fish
NP -> tanks
NP -> rods
```

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
~~VP → V NP PP~~
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

把RHS有三項的改成兩項
VP –> V @VP_V
@VP_V –> NP PP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → *e*
PP → P NP

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

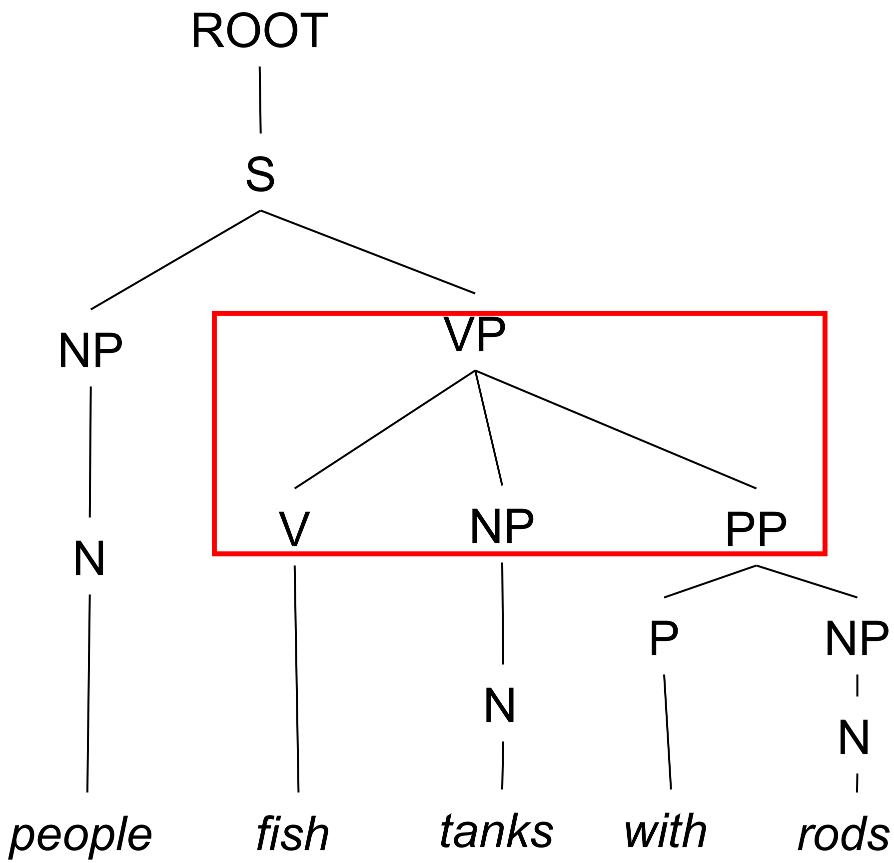
Chomsky Normal Form

- You should think of this as a transformation for efficient parsing
- With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
- In practice full Chomsky Normal Form is a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries/empties is trickier

- **Binarization** is crucial for cubic time CFG parsing

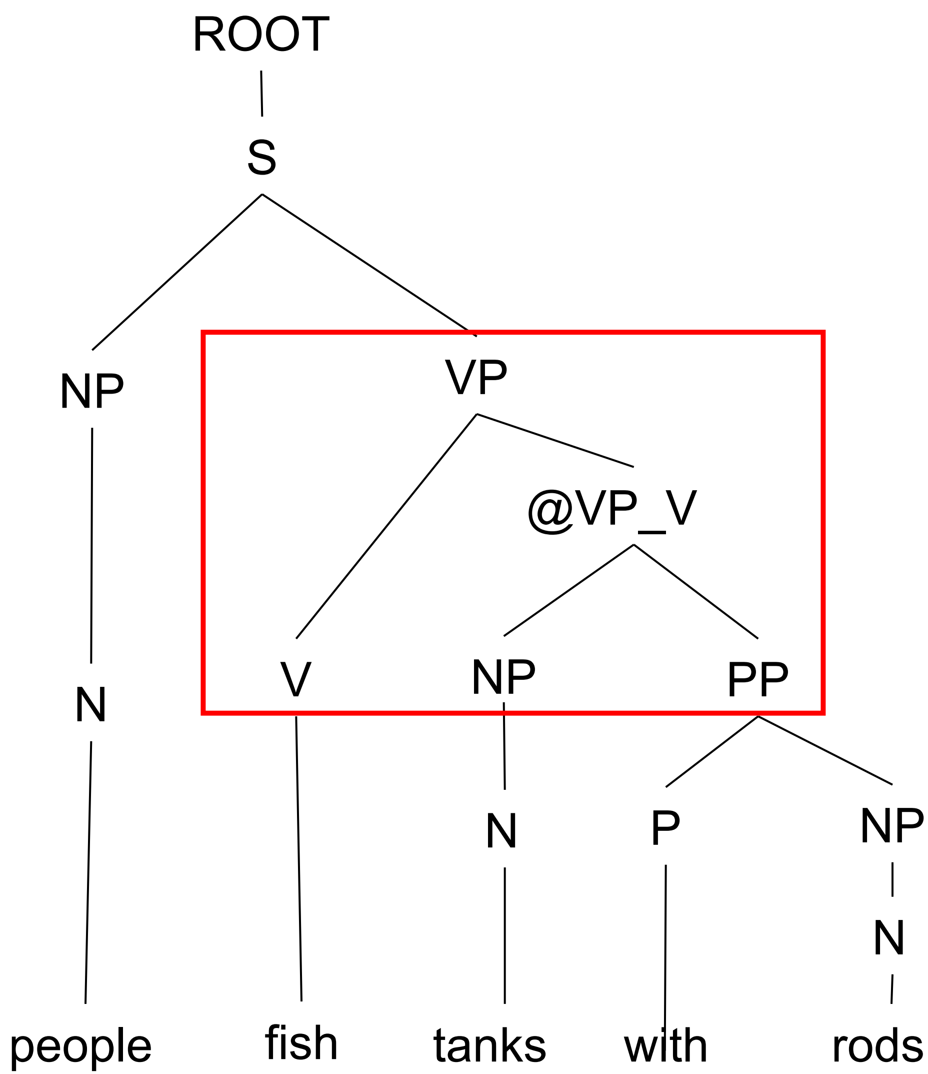
- The rest isn't necessary; it just makes the algorithms cleaner and a bit quicker

An example: before binarization…



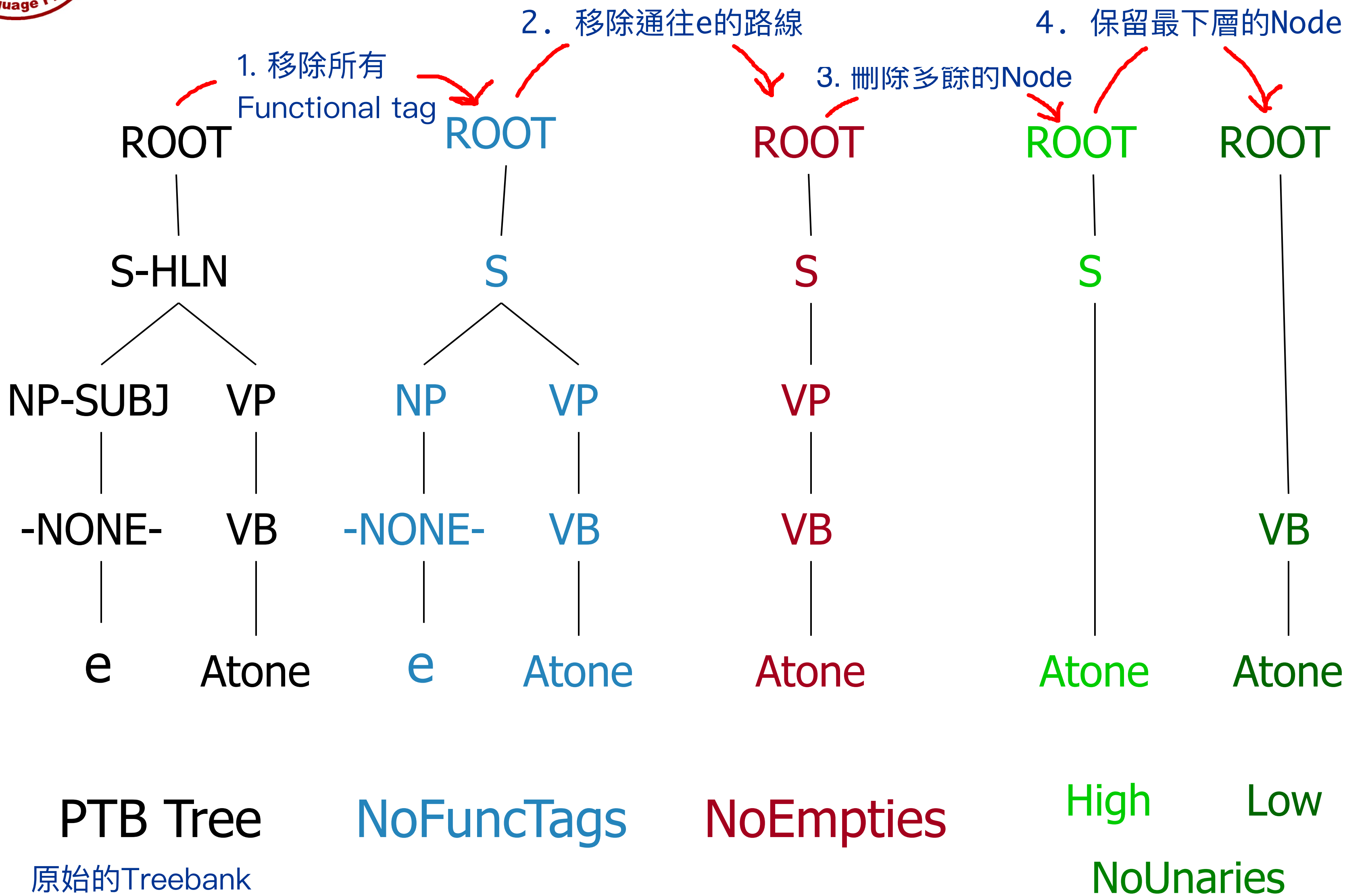
Subtree with 3 or more children
這種文法就需要被修改

After binarization...

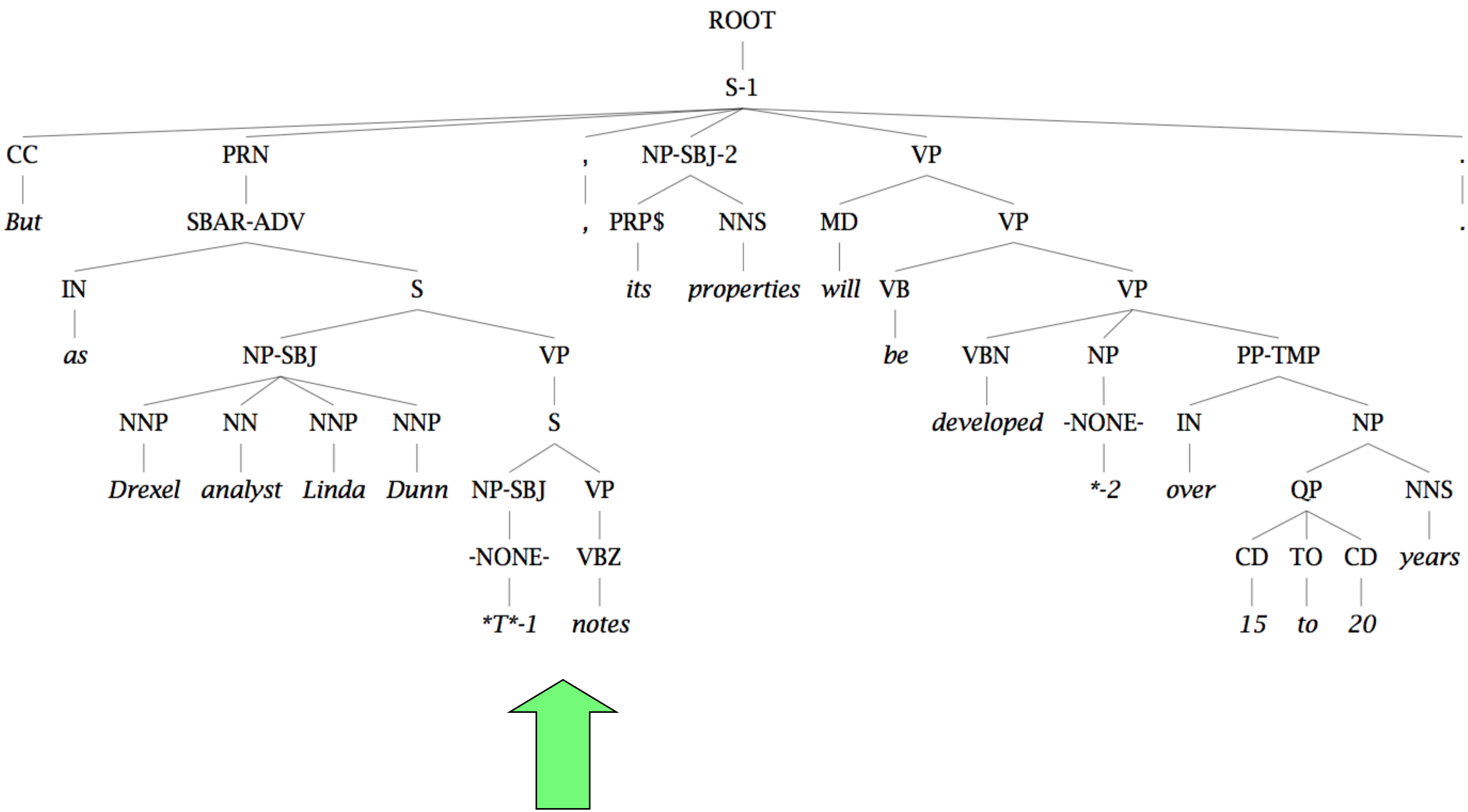


Christopher Manning

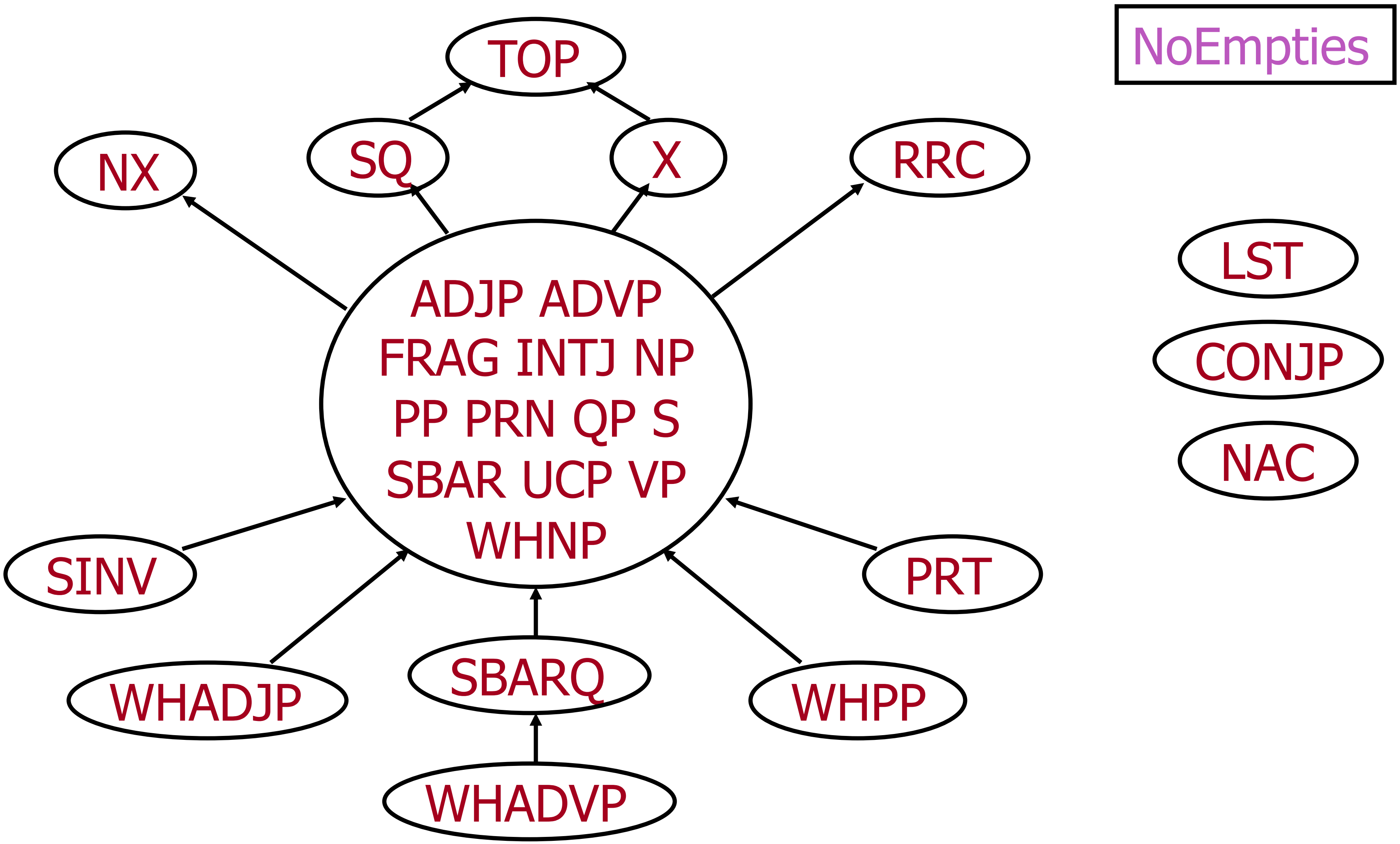
Treebank: empties and unaries



Unary rules: alchemy in the land of treebanks



Same-Span Reachability



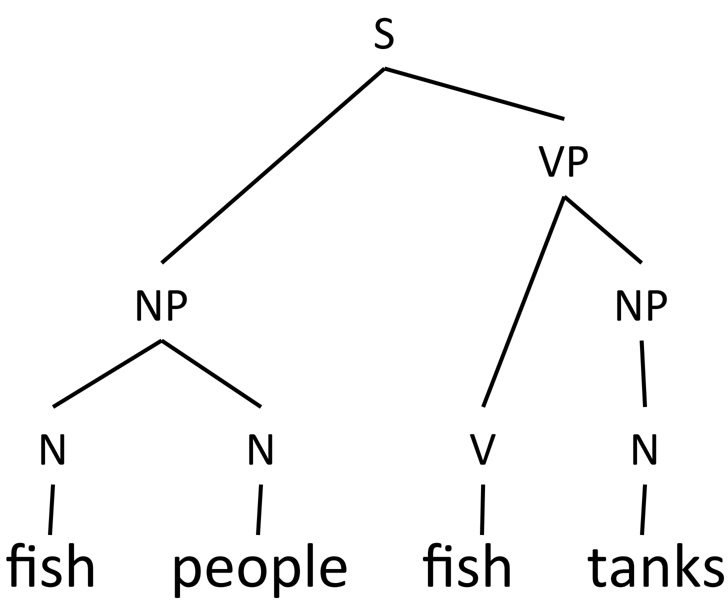
Grammar Transforms

Restricting the grammar form for efficient parsing

CKY Parsing

Exact polynomial time parsing of (P)CFGs

Constituency Parsing

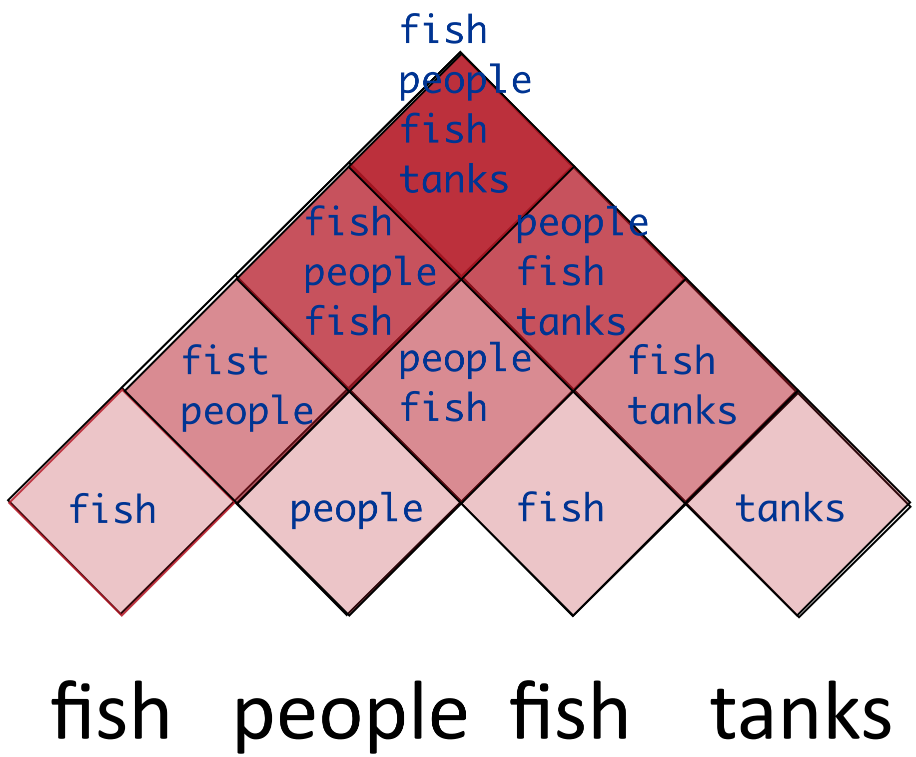


PCFG

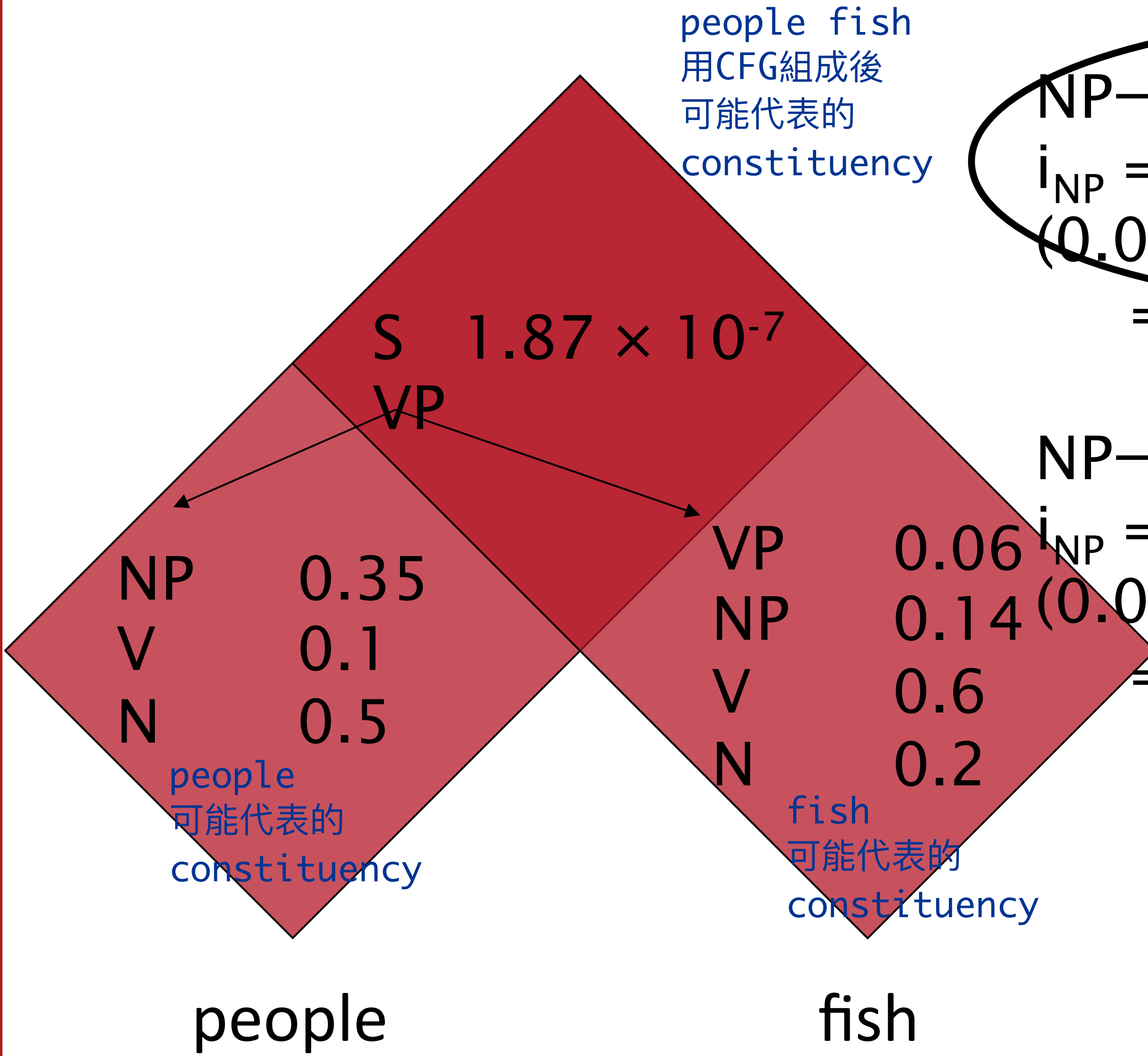
Rule	Prob θ_i
S → NP VP	θ_0
NP → NP NP	θ_1
…	
N → fish	θ_{42}
N → people	θ_{43}
V → fish	θ_{44}
…	

Cocke-Kasami-Younger (CKY) Constituency Parsing

Parse Triangle / Chart



Viterbi (Max) Scores



$NP \rightarrow NN\ NNS$ 0.13

$i_{NP} = (0.13)(0.0023)(0.0014)$

$= 1.87 \times 10^{-7}$

$NP \rightarrow NNP\ NNS$ 0.056

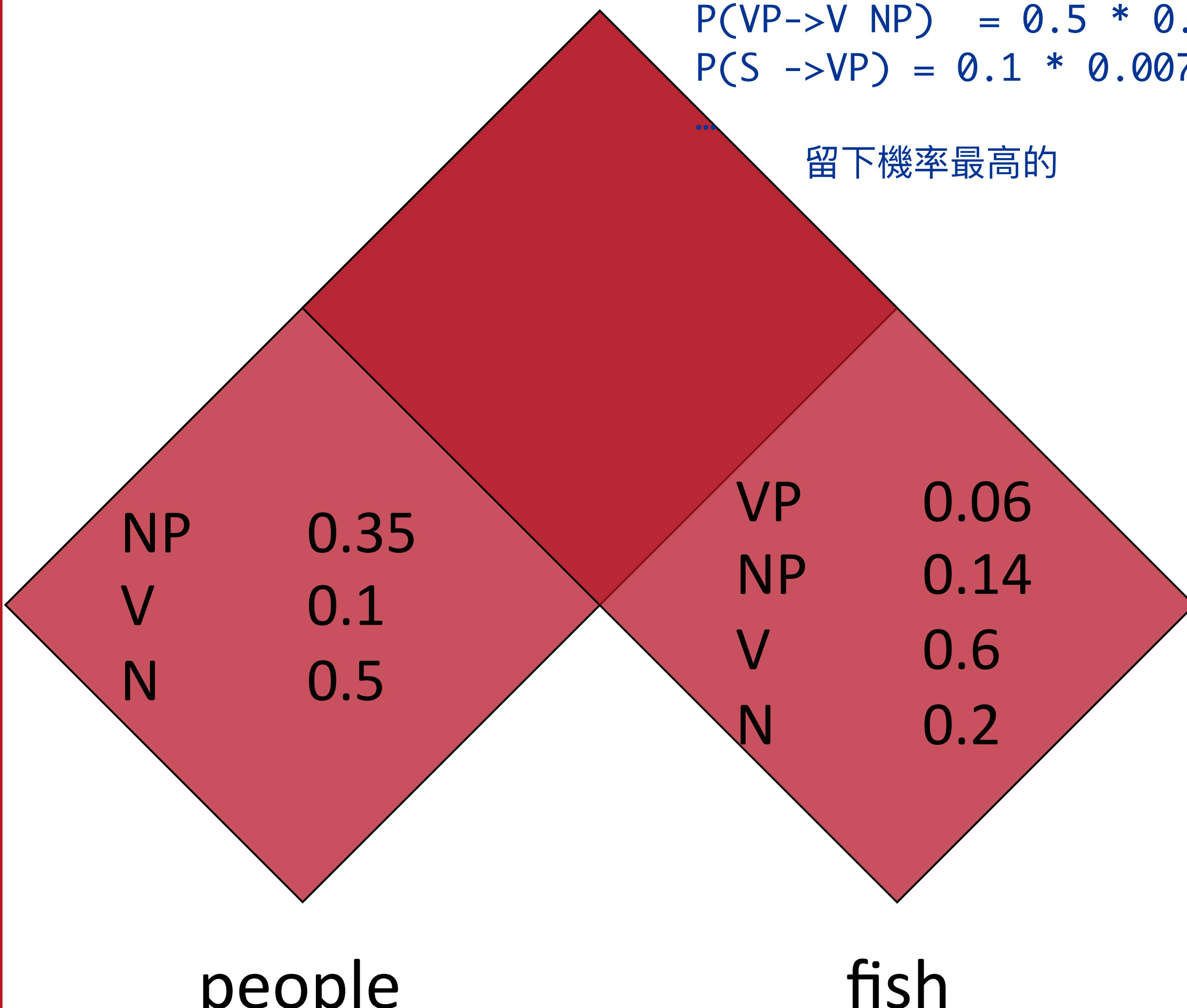
$i_{NP} = (0.056)(0.001)(0.0014)$

$= 7.84 \times 10^{-8}$

Viterbi (Max) Scores

```
P(NP->NP NP) = 0.1 * 0.35 * 0.14
P(VP->V NP)  = 0.5 * 0.1  * 0.14 = 0.007
P(S ->VP) = 0.1 * 0.007
...
```

留下機率最高的



S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn't increase algorithmic complexity
- Empties can be incorporated
  - Use fenceposts 把s加到句子最前面，所以那張Chart的邊長就會是n+1
  - Doesn't increase complexity; essentially like unaries
- Binarization is *vital*
  - Without binarization, you don't get parsing cubic in the length of the sentence and in the number of nonterminals in the grammar
    - Binarization may be an explicit transformation or implicit in how the parser works (Early-style dotted rules), but it's always there.

The CKY algorithm (1960/1965) … extended to unaries

```
function CKY(words, grammar) returns [most_probable_parse,prob]
  score = new double[#(words)+1][#(words)+1][#(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
```

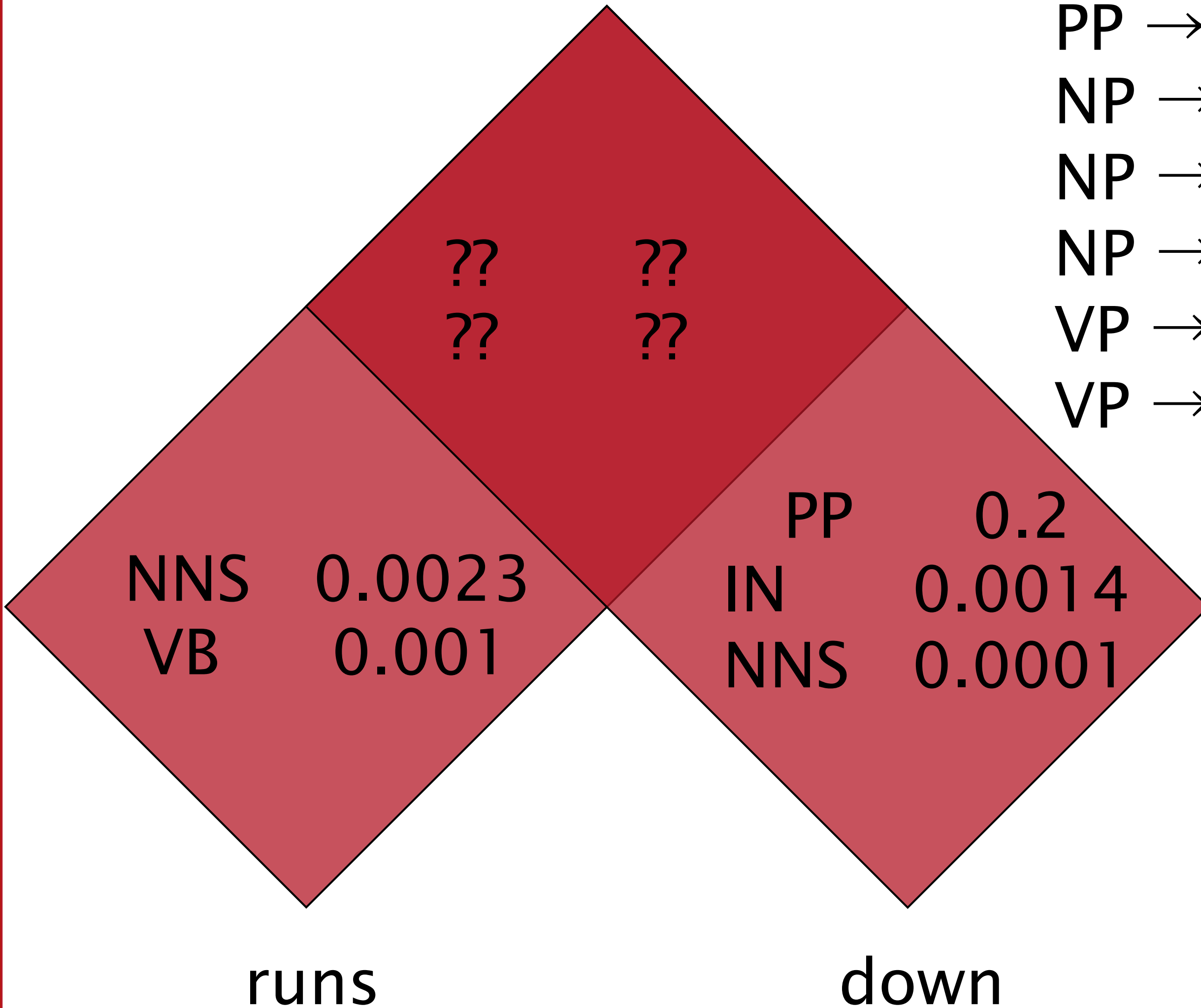
A -> terminal 0.3

unary rules

The CKY algorithm (1960/1965) … extended to unaries

```
for span = 2 to #(words) start from length >= 2
  for begin = 0 to #(words)- span 從左到右
    end = begin + span
    for split = begin+1 to end-1
      for A,B,C in nonterms
binary  prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
rules   if prob > score[begin][end][A]
          score[begin]end][A] = prob
          back[begin][end][A] = new Triple(split,B,C)
    //handle unaries
    boolean added = true
    while added
      added = false               unaries rules
      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)
```

Quiz Question!



PP → IN	0.002
NP → NNS NNS	0.01
NP → NNS NP	0.005
NP → NNS PP	0.01
VP → VB PP	0.045
VP → VB NP	0.015

What constituents (with what probability can you make?

CKY Parsing

Exact polynomial time parsing of (P)CFGs

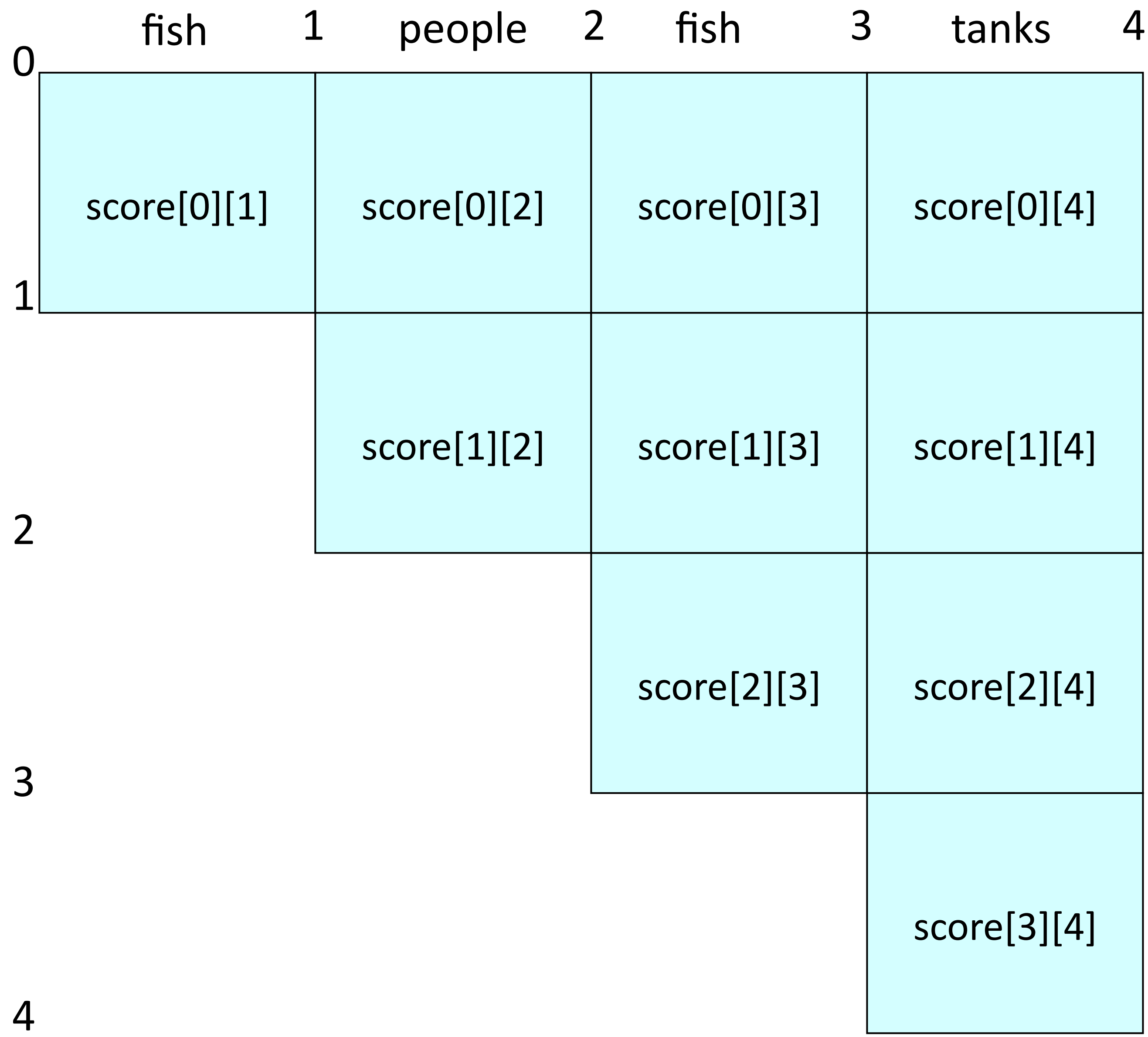
CKY Parsing

A worked example

The grammar: Binary, no epsilons,

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0



0
fish
1
people
2
fish
3
tanks
4
score[0][1]
score[0][2]
score[0][3]
score[0][4]
1
score[1][2]
score[1][3]
score[1][4]
2
score[2][3]
score[2][4]
3
score[3][4]
4

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

總之長度為一的
直接查表填入就可以了

```
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]);
```

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish (0–1)	people (1–2)	fish (2–3)	tanks (3–4)
0	N → fish 0.2 V → fish 0.6 S->VP = 0.006 VP->V = 0.06 NP->N = 0.14			
1		N → people 0.5 V → people 0.1		
2			N → fish 0.2 V → fish 0.6	
3				N → tanks 0.2 V → tanks 0.1

對於長度為1的
遞迴向前改寫所有grammar
rules的可能

```
// 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
```

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	S->NP VP = 0.9*0.14*0.01 VP->V NP = 0.5*0.6*0.35 NP->NP NP = 0.1*0.14*0.35		
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001		
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

fish和people
都有可能是
N, V, NP, VP, S找出所有可能
向前改寫的排列組合，並算機率

```
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)
```

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → NP VP 0.00126		
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → NP VP 0.00378
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

S->VP
=0.105*0.1>0.00126
所以把原本的S->NP VP换掉

```
//handle unaries
boolean added = 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
```

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105		
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

fish people fish有兩種解讀法，分別是
[fish people] fish (紅框)
fish [people fish] (綠框)
所以要列出這兩組框的所有可能排列組合

```
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)
```

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003

這邊要像49頁一樣
找出unaries
然後看看能不能取代原本的
例如S–>VP =
0.1*0.00147<0.000882
無法取代

```
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)
```

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

這邊要像49頁一樣
找出unaries
然後看看能不能取代原本的
例如S–>VP =
0.1*0.000098<0.000882
無法取代

```
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)
```

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

這邊就像50頁那樣
fish people fish tanks
有以下三種解讀方法
[fish people fish] [tanks]（紅框）
[fish people] [fish tanks]（綠框）
[fish] [people fish tanks]（紫框）
一樣列出所有排列組合後，把機率最高的填入
最後得到這句話的最佳排列為S–>NP VP
問題來了，TREE呢？？？？
哪一塊代表NP，哪一塊代表VP？
所以要呼叫最後一行的buildTree，進行backtracking
就能找到路徑，畫出最終的樹

Call buildTree(score, back) to get the best parse

CKY Parsing

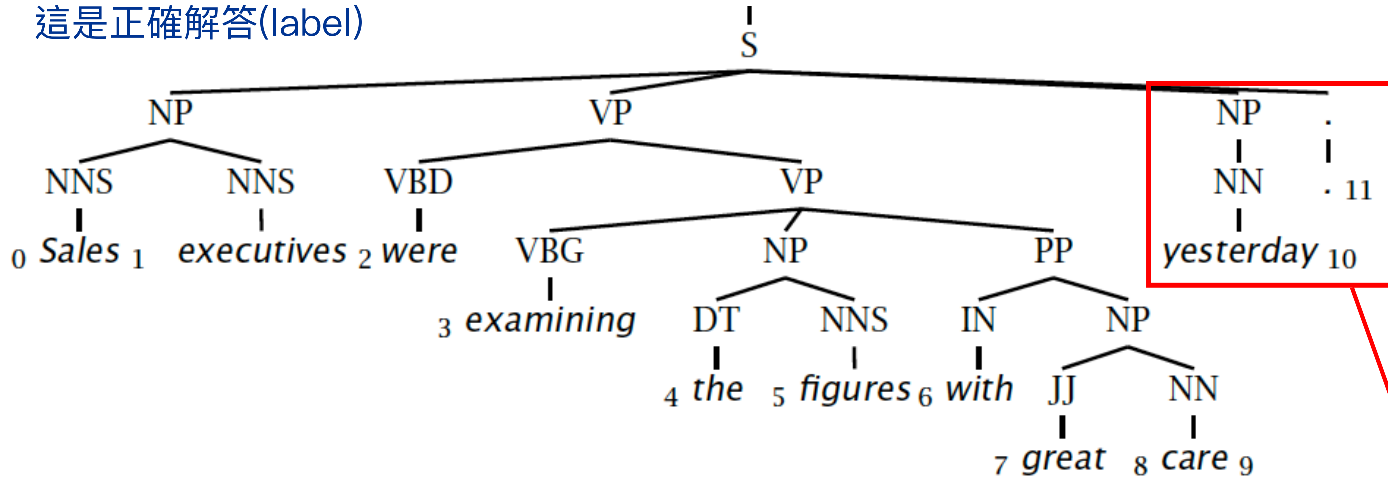
A worked example

Constituency Parser Evaluation

Evaluating constituency parsing

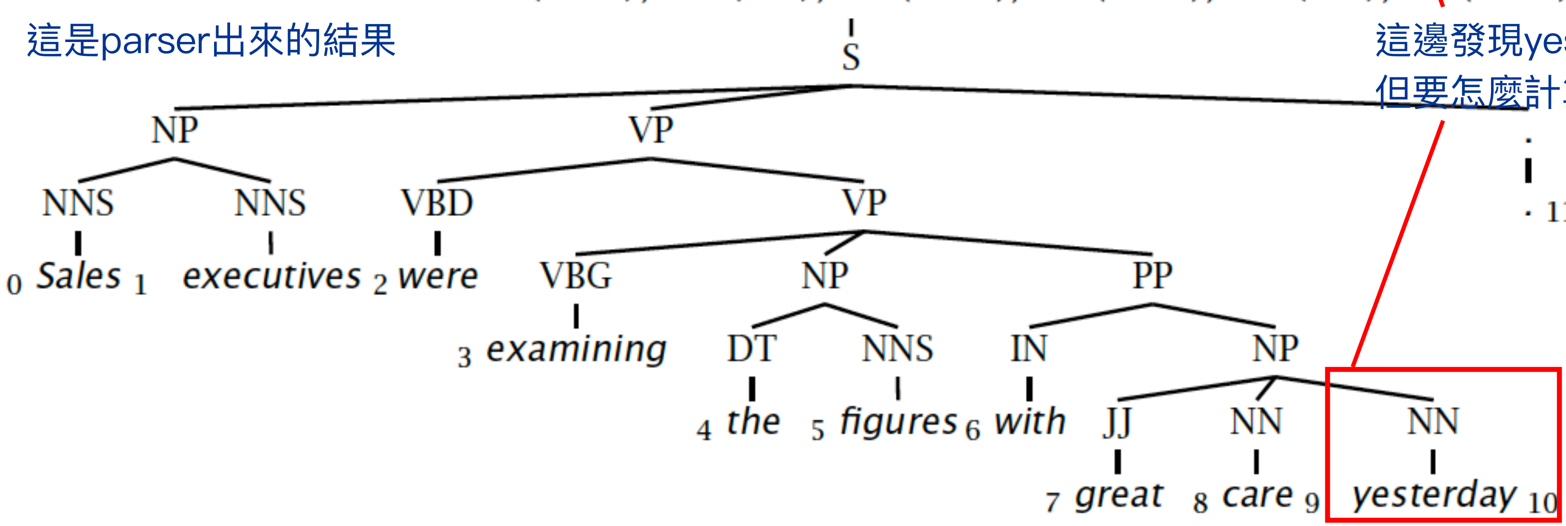
Gold standard brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)

這是正確解答(label)



Candidate brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)

這是parser出來的結果



這邊發現yesterday錯了
但要怎麼計算他的error

Evaluating constituency parsing

Gold standard brackets:

S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)

Candidate brackets:

S-(0:11), **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)

Labeled Precision	3/7 = 42.9%	找到的7個有3個是全對的
Labeled Recall	3/8 = 37.5%	正解的8個有3個被找到了
LP/LR F1	40.0%	
Tagging Accuracy	11/11 = 100.0%	總共11個字各自都標對了

缺點：像這個例子只錯了一個yesterday
但因為會導致其他tag的位置錯很多
所以效能會比預期低很多

How good are PCFGs?

- Penn WSJ parsing accuracy: about 73% LP/LR F1
- Robust
  - Usually admit everything, but with low probability
- Partial solution for grammar ambiguity
  - A PCFG gives some idea of the plausibility of a parse
  - But not so good because the independence assumptions are too strong
- Give a probabilistic language model
  - But in the simple case it performs worse than a trigram model
- The problem seems to be that PCFGs lack the lexicalization of a trigram model

Constituency Parser Evaluation