Probabilistic Context-free Grammars

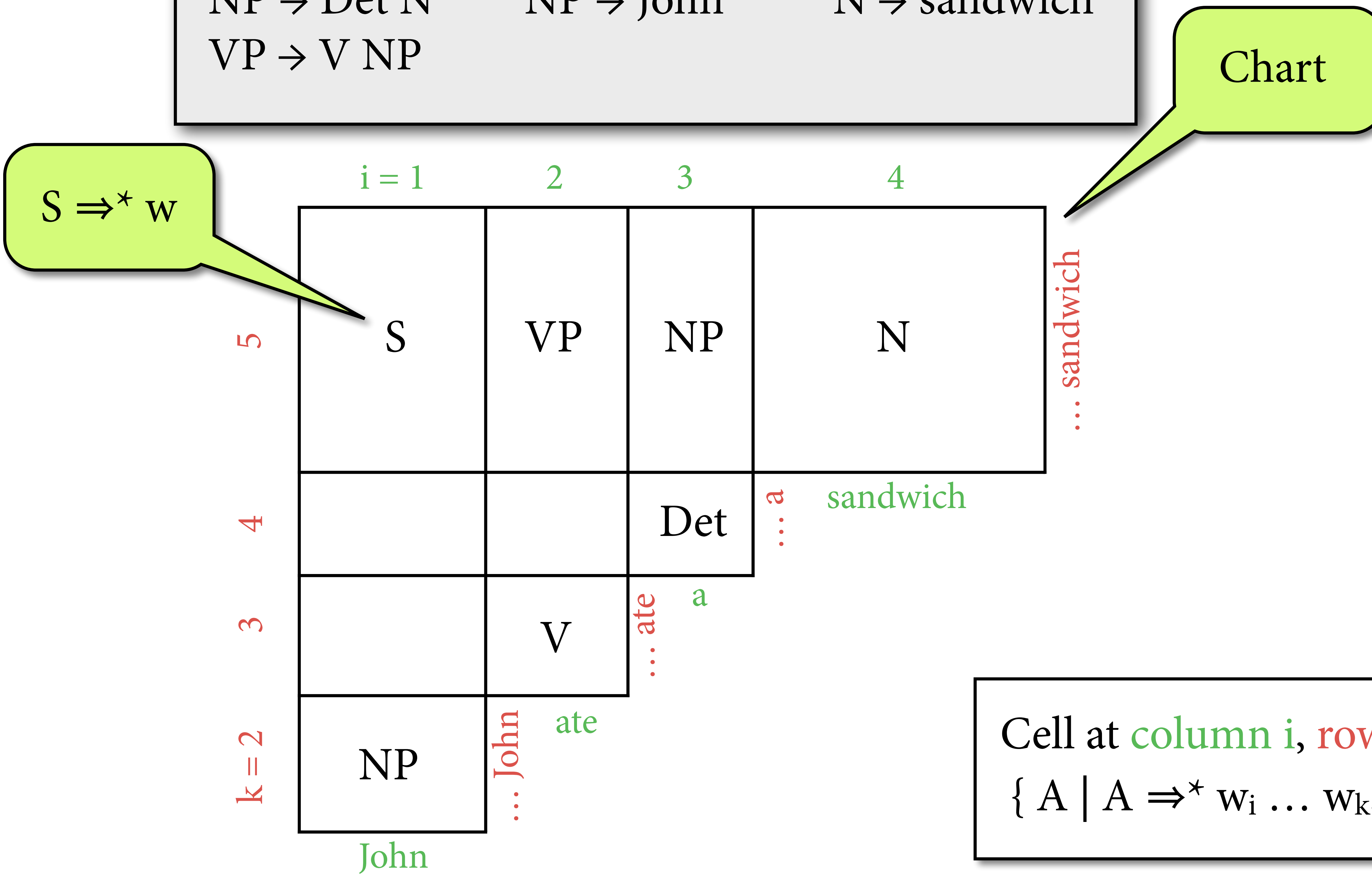
Computational Linguistics

Alexander Koller

23 November 2018

The CKY Recognizer

S → NP VP	V → ate	Det → a
NP → Det N	NP → John	N → sandwich
VP → V NP		



Cell at column i, row k:

$\{ A \mid A \Rightarrow^* w_i \ldots w_{k-1} \}$

CKY recognizer: pseudocode

Data structure: $Ch(i,k)$ eventually contains $\{A \mid A \Rightarrow^* w_i \ldots w_{k-1}\}$ (initially all empty).

```
for each i from 1 to n:
    for each production rule A → wi:
        add A to Ch(i, i+1)

for each width b from 2 to n:
    for each start position i from 1 to n-b+1:
        for each left width k from 1 to b-1:
            for each B ∈ Ch(i, i+k) and C ∈ Ch(i+k,i+b):
                for each production rule A → B C:
                    add A to Ch(i,i+b)

claim that w ∈ L(G) iff S ∈ Ch(1,n+1)
```

Complexity

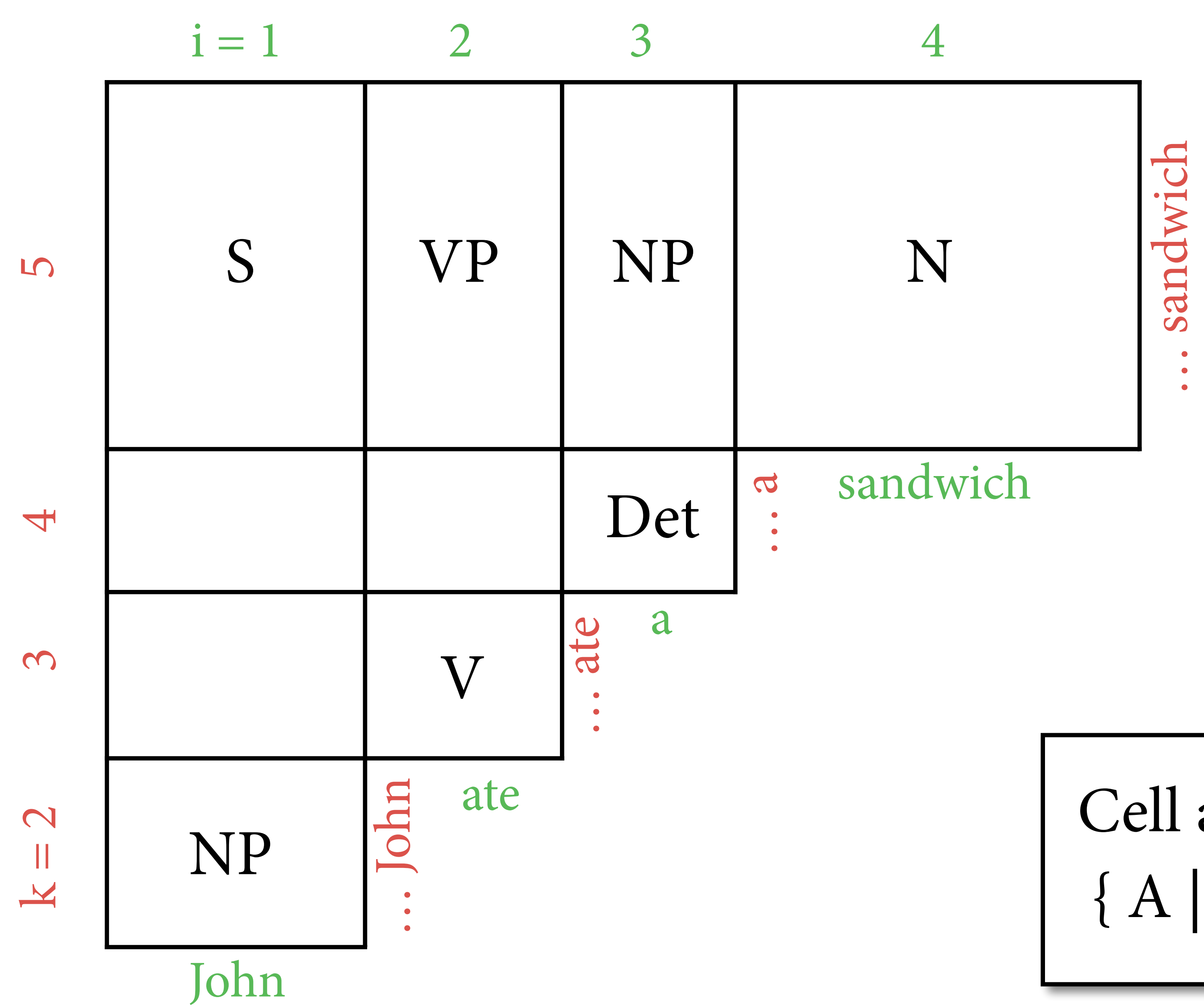
- *Time* complexity of CKY recognizer is $O(n^3)$, although number of parse trees grows exponentially.
- *Space* complexity of CKY recognizer is $O(n^2)$ (one cell for each substring).
- Efficiency depends crucially on CNF.
Naive generalization of CKY to rules $A \rightarrow B_1 \ldots B_r$ raises time complexity to $O(n^{r+1})$.

Recognizer to Parser

S → NP VP
NP → Det N
VP → V NP

V → ate
NP → John

Det → a
N → sandwich



Cell at column i, row k:

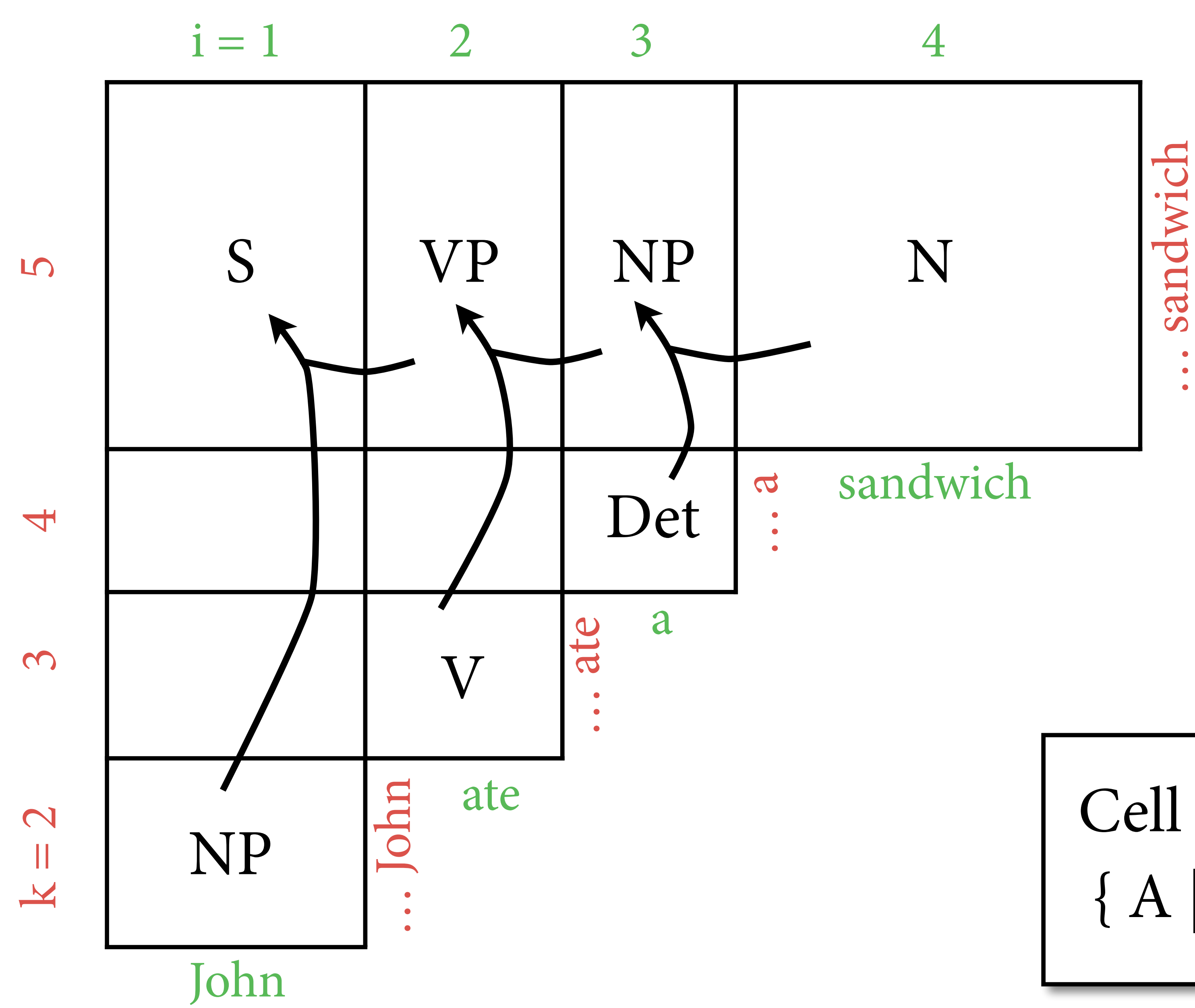
$\{ A \mid A \Rightarrow^{*} w_i \ldots w_{k-1} \}$

Recognizer to Parser

S → NP VP
NP → Det N
VP → V NP

V → ate
NP → John

Det → a
N → sandwich



Cell at column i, row k:

$\{ A \mid A \Rightarrow^* w_i \ldots w_{k-1} \}$

Recognizer to Parser

- Parser: need to construct parse trees from chart.
- Do this by memorizing how each $A \in Ch(i,k)$ can be constructed from smaller parts.
  - built from $B \in Ch(i,j)$ and $C \in Ch(j,k)$ using $A \to B\ C$: store (B,C,j) in *backpointer* for A in $Ch(i,k)$.
  - analogous to backpointers in HMMs
- Once chart has been filled, enumerate trees recursively by following backpointers, starting at $S \in Ch(1,n+1)$.

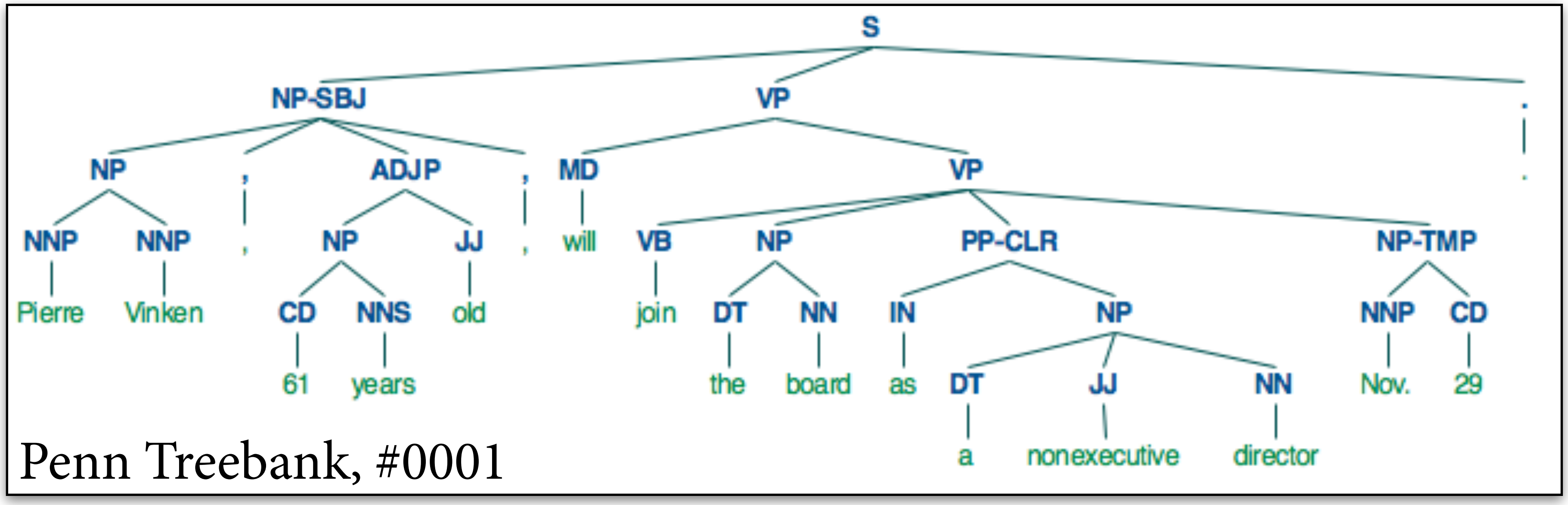
Let's play a game

- Given a nonterminal symbol, expand it.
- You can take one of two moves:
  - expand nonterminal into a sequence of other nontermianls
  - use nonterminals S, NP, VP, PP, … or POS tags
  - expand nonterminal into a word

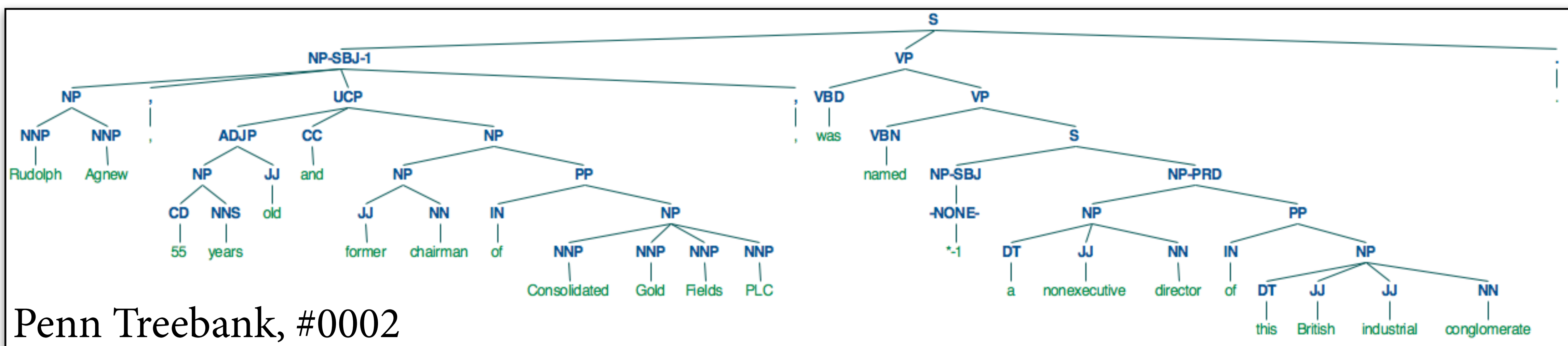
Penn Treebank POS tags

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Some real trees



Penn Treebank, #0001

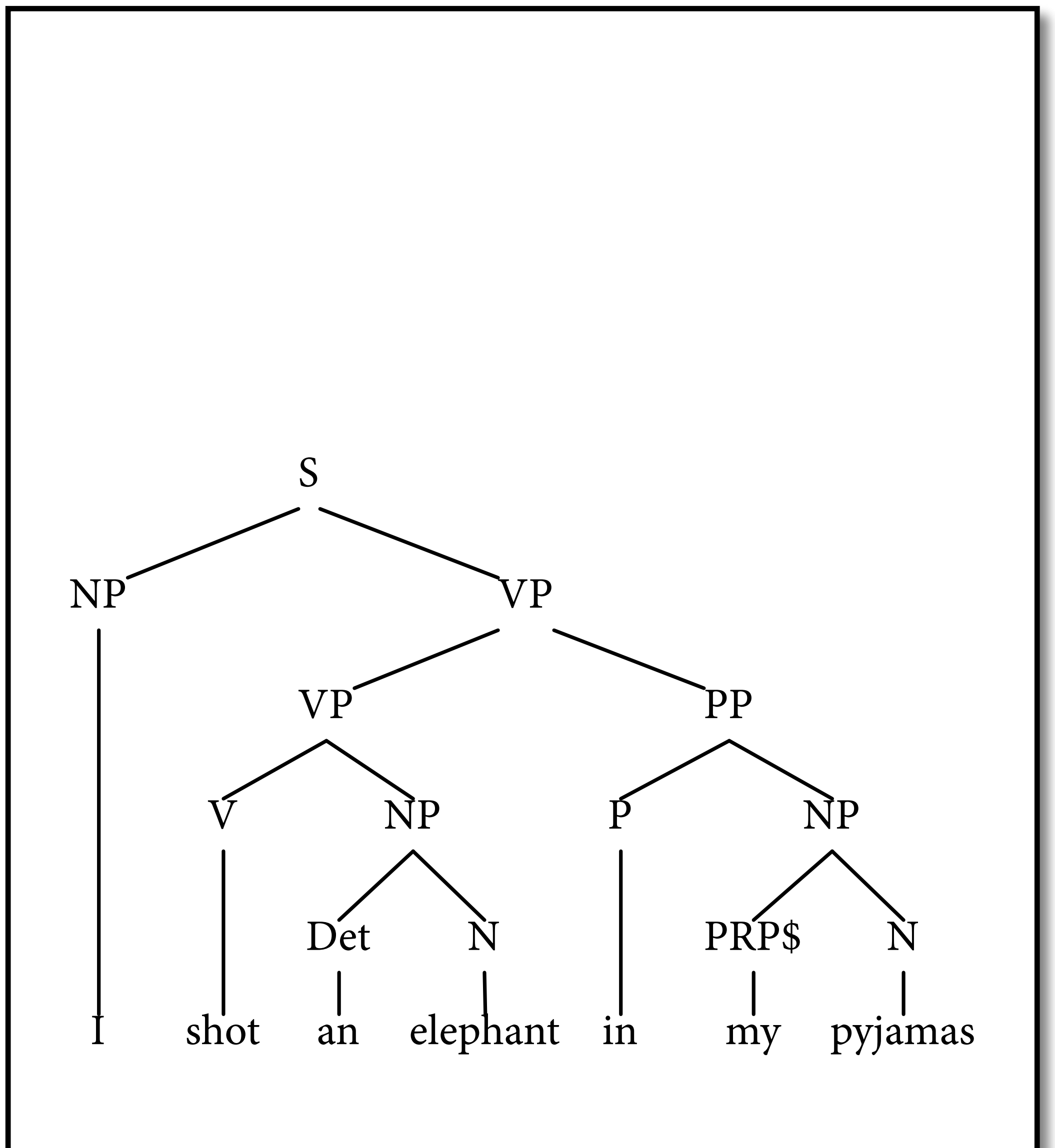


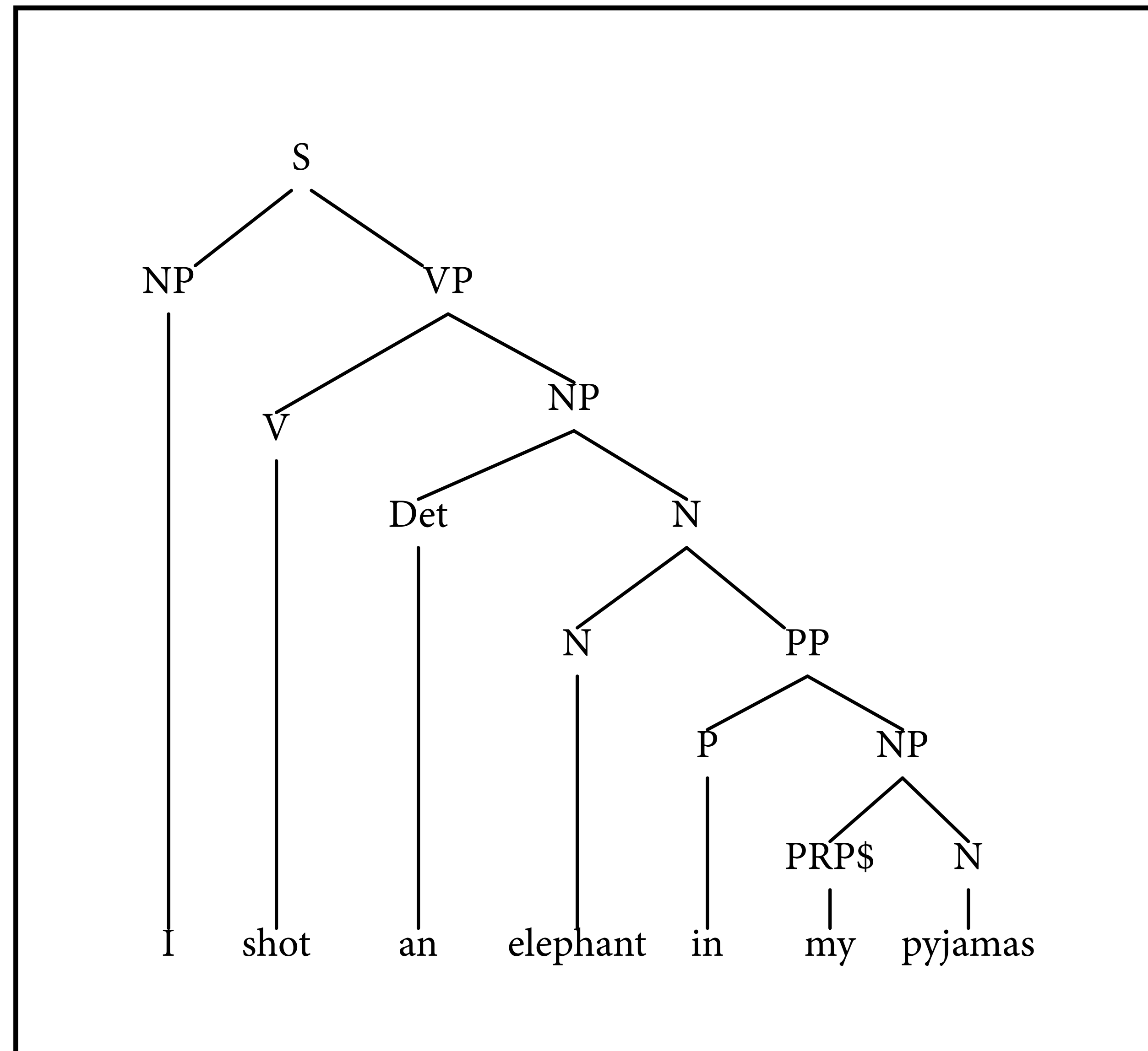
Penn Treebank, #0002

```
nltk.corpus.treebank.parsed_sents("wsj_0001.mrg")[0].draw()
```

Ambiguity

Need to *disambiguate:* find “correct” parse tree for ambiguous sentence.





How do we identify the “correct” tree?
How do we compute it efficiently? (Remember: exponential number of readings.)

Probabilistic CFGs

- A *probabilistic context-free grammar (PCFG)* is a context-free grammar in which
  - each production rule A → w has a probability P(A → w | A): when we expand A, how likely is it that we choose A → w?
  - for each nonterminal A, probabilities must sum to one:

    $$\sum_{w} P(A \to w \mid A) = 1$$

  - we will write P(A → w) instead of P(A → w | A) for short

An example

S → NP VP	[1.0]	VP → V NP	[0.5]
NP → Det N	[0.8]	VP → VP PP	[0.5]
NP → i	[0.2]	V → shot	[1.0]
N → N PP	[0.4]	PP → P NP	[1.0]
N → elephant	[0.3]	P → in	[1.0]
N → pyjamas	[0.3]	Det → an	[0.5]
		Det → my	[0.5]

(let's pretend for simplicity that Det = PRP$)

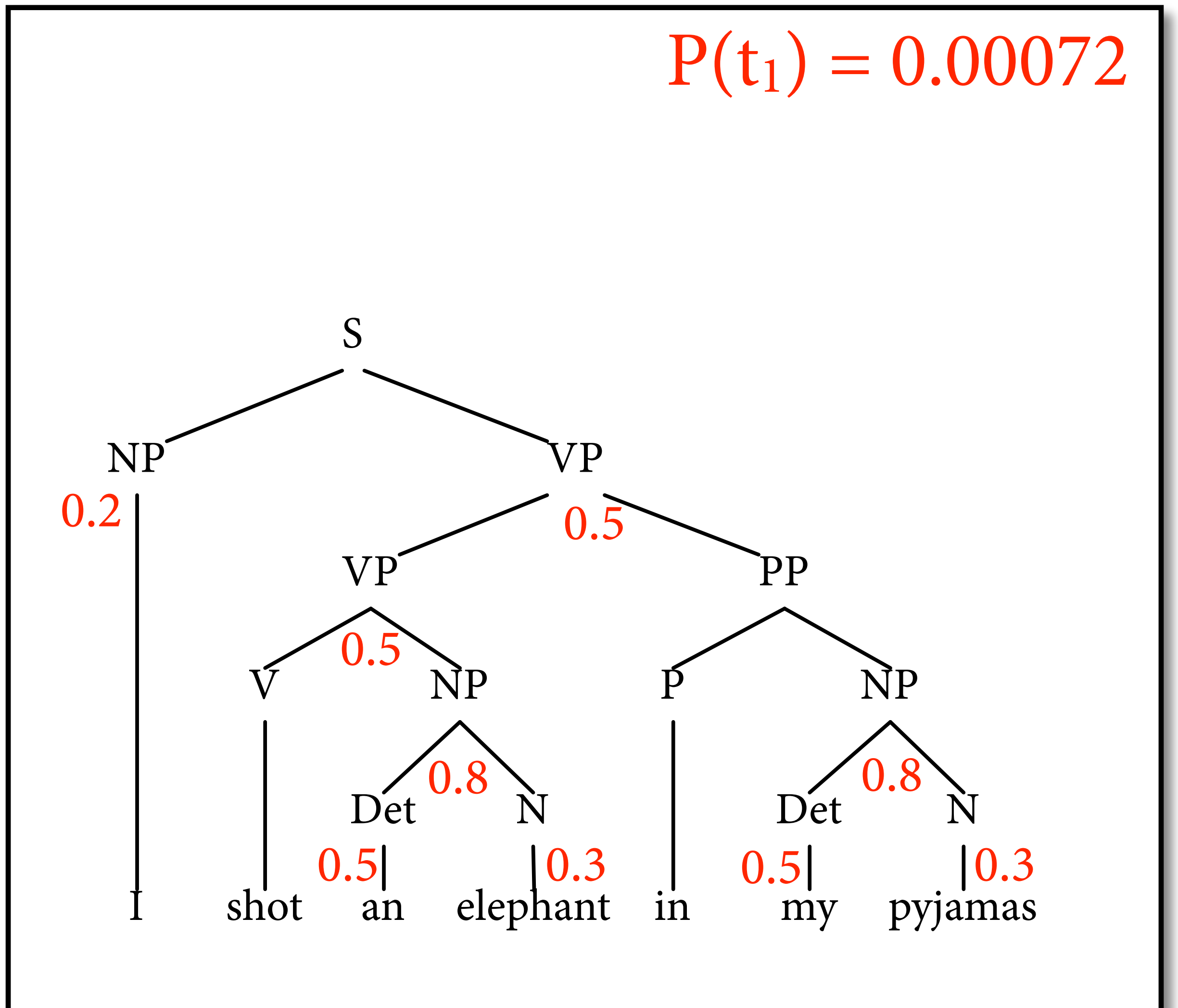
Generative process

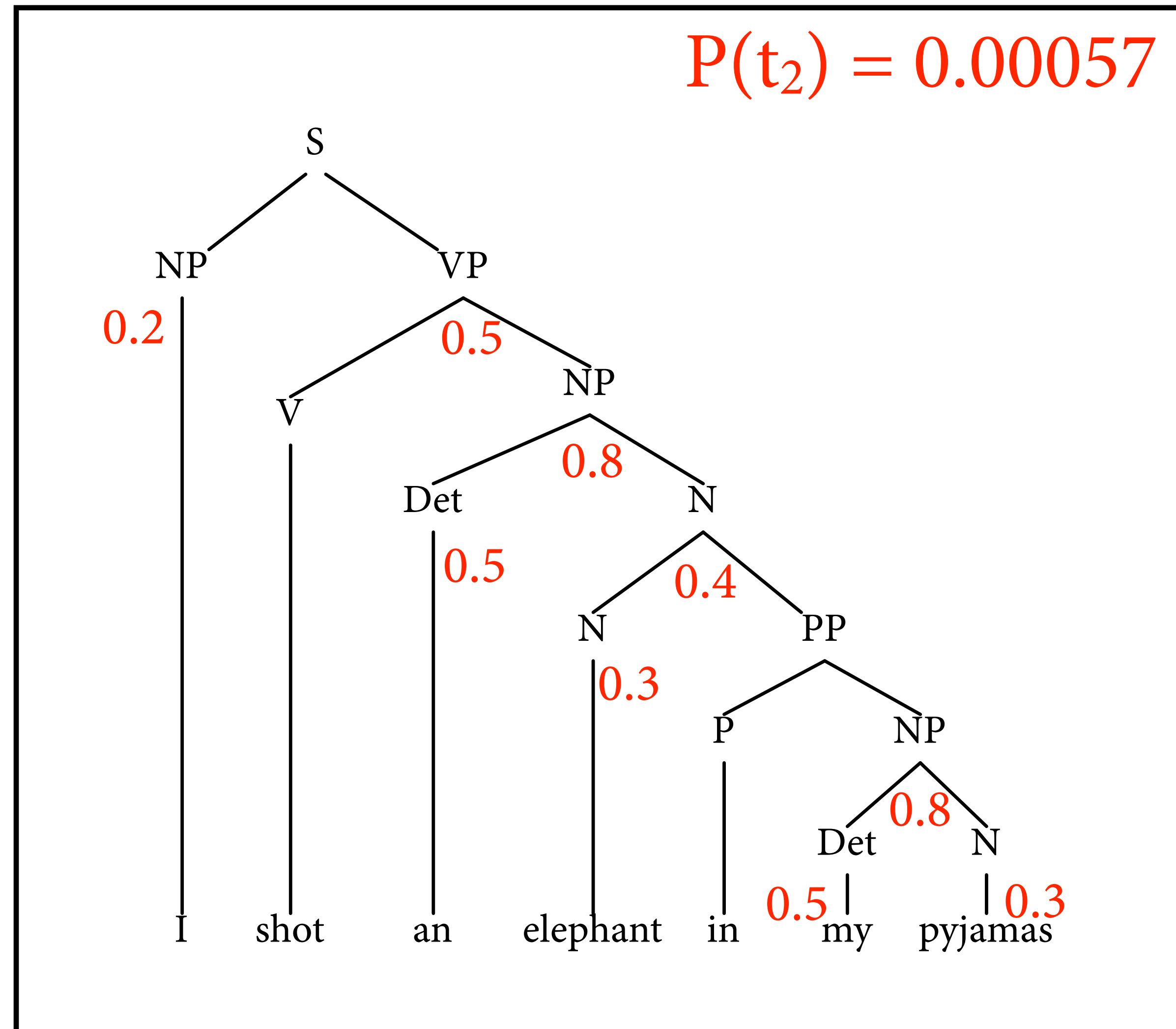
- PCFG generates random derivations of CFG.
  - each event (expand nonterminal by production rule) statistically independent of all the others

$$S \overset{1.0}{\Rightarrow} NP\ VP \overset{0.2}{\Rightarrow} i\ VP \overset{0.5}{\Rightarrow} i\ VP\ PP \Rightarrow^* \text{i shot an elephant in my pyjamas} \quad 0.00072$$

$$S \overset{1.0}{\Rightarrow} NP\ VP \overset{0.2}{\Rightarrow} i\ VP \overset{0.4}{\Rightarrow^*} i\ V\ Det\ N \underset{0.4}{\Rightarrow} i\ V\ Det\ N\ PP \Rightarrow^* \text{i shot} \ldots \text{pyjamas} \quad 0.00057$$

Parse trees





"correct" = more probable parse tree

Language modeling

- As with other generative models (HMMs!), can define probability P(w) of string by marginalizing over its possible parses:

$$P(w) = \sum_{t \in \mathsf{parses}(w)} P(t)$$

- Can compute this efficiently with *inside probabilities*, see next time.

Disambiguation

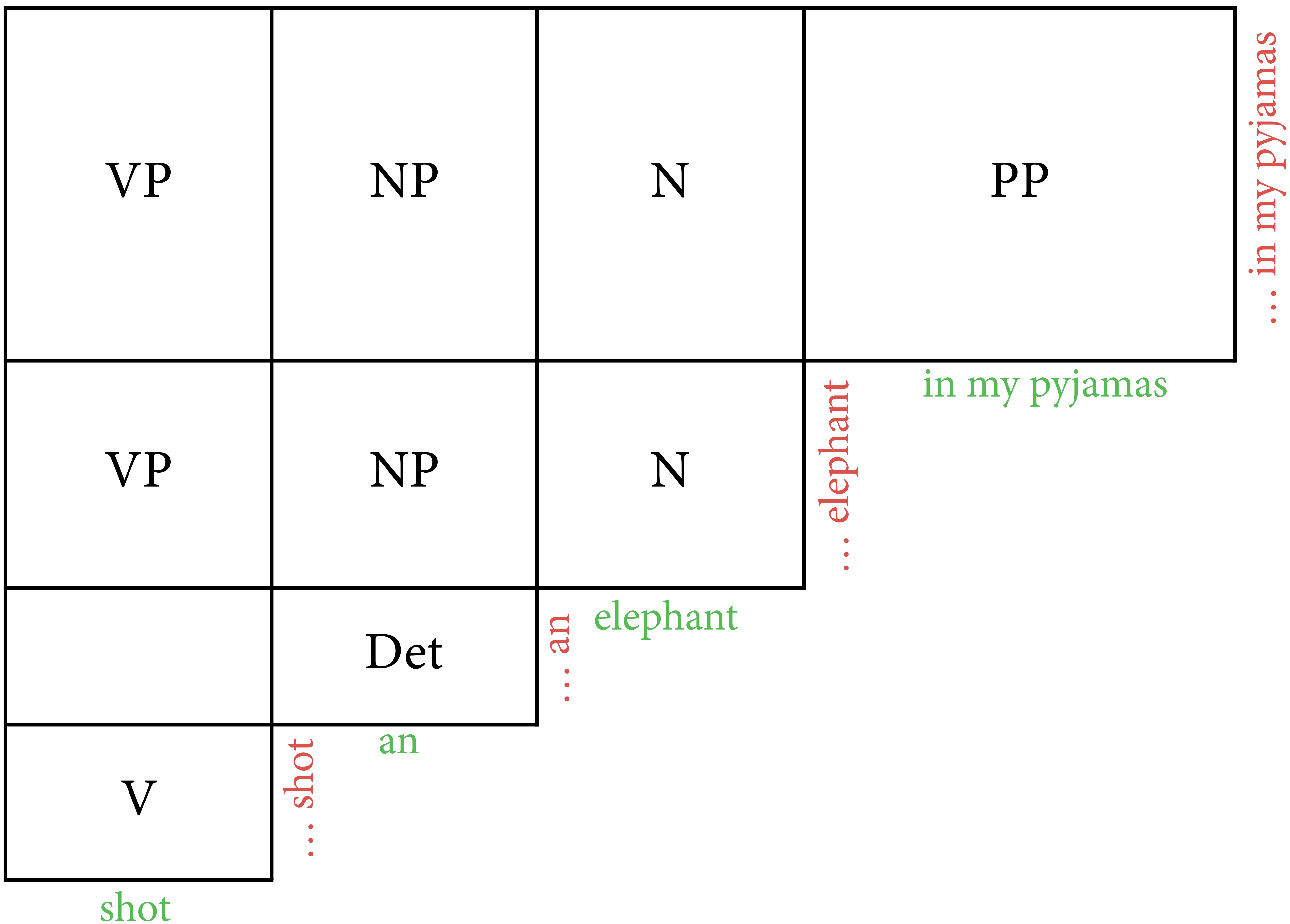
- Assumption: "correct" parse tree = the parse tree that had highest prob of being generated by random process, i.e. $\underset{t \in \mathsf{parses(w)}}{\operatorname{argmax}} P(t)$

- We use a variant of the Viterbi algorithm to compute it.

- Here, Viterbi based on CKY; can do it with other parsing algorithms too.

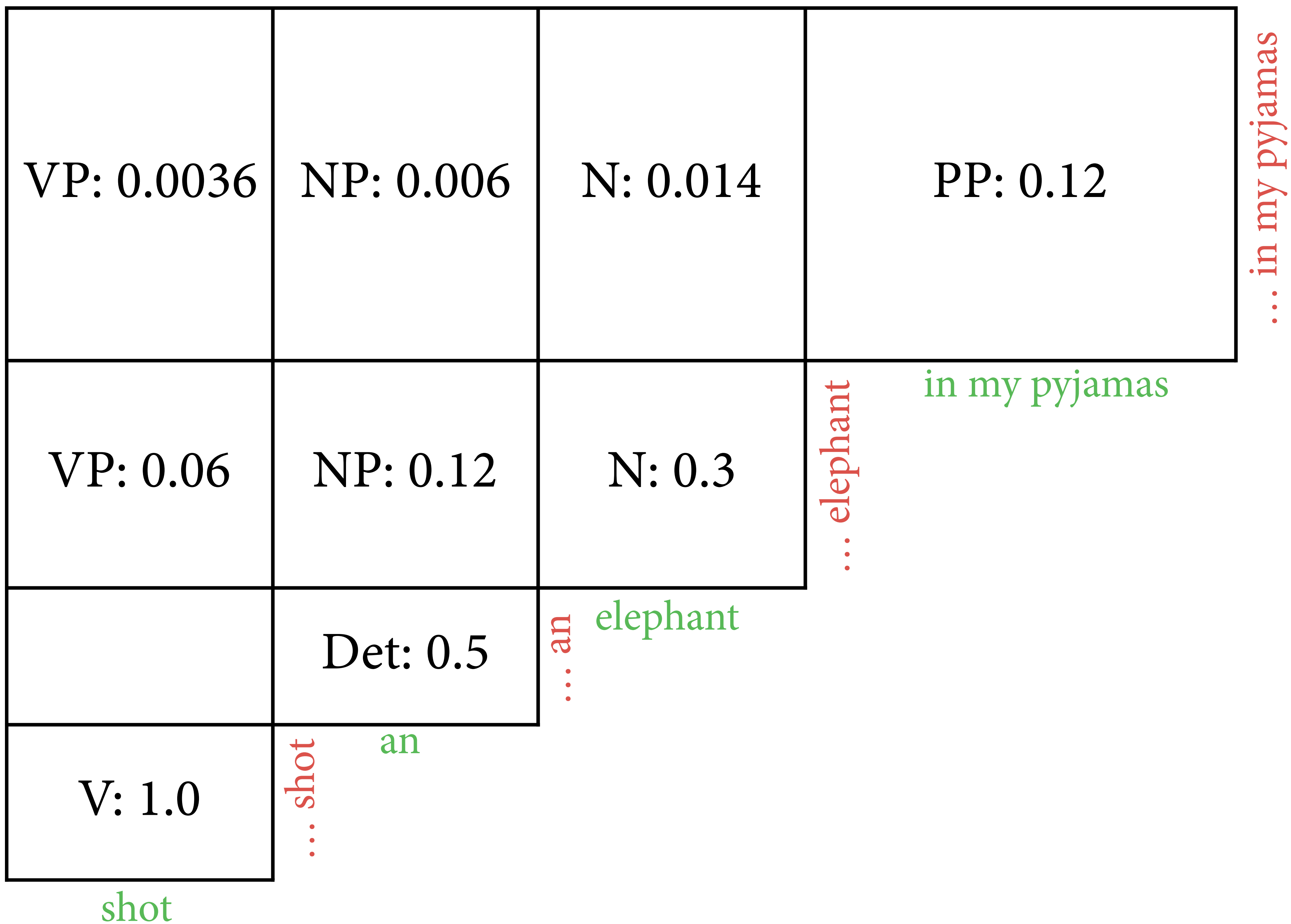
The intuition

Ordinary CKY parse chart: $Ch(i,k) = \{A \mid A \Rightarrow^* w_i \ldots w_{k-1}\}$



The intuition

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d: A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



Viterbi CKY

- Define for each span (i,k) and each nonterminal A the probability

$$V(A,i,k) = \max_{A \overset{d}{\Rightarrow}^* w_i \ldots w_{k-1}} P(d)$$

- Compute V iteratively “bottom up”, i.e. starting from small spans and working our way up to longer spans.

$$V(A,i,i+1) = P(A \rightarrow w_i)$$

$$V(A,i,k) = \max_{\substack{A \rightarrow B\ C \\ i<j<k}} P(A \rightarrow B\ C) \cdot V(B,i,j) \cdot V(C,j,k)$$

Viterbi CKY - pseudocode

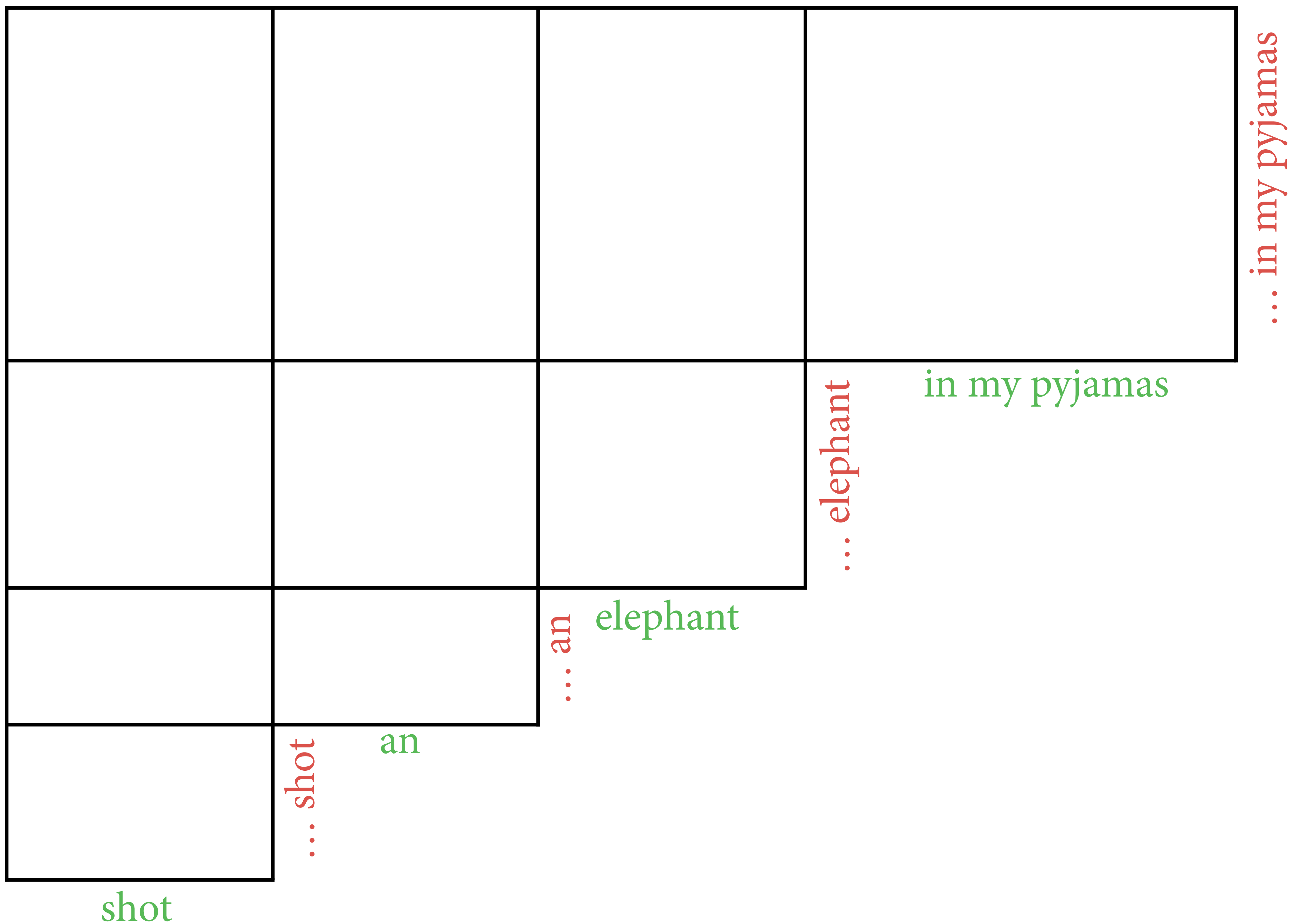
```
set all V[A,i,j] to 0

for all i from 1 to n:
    for all A with rule A -> wi:
        add A to Ch(i,i+1)
        V[A,i,i+1] = P(A -> wi)

for all b from 2 to n:
  for all i from 1 to n-b+1:
    for all k from 1 to b-1:
      for all B in Ch(i,i+k) and C in Ch(i+k,i+b):
        for all production rules A -> B C:
          add A to Ch(i,i+b)
          if P(A -> B C) * V[B,i,i+k] * V[C,i+k,i+b] > V[A,i,i+b]:
            V[A,i,i+b] = P(A -> B C) * V[B,i,i+k] * V[C,i+k,i+b]
```

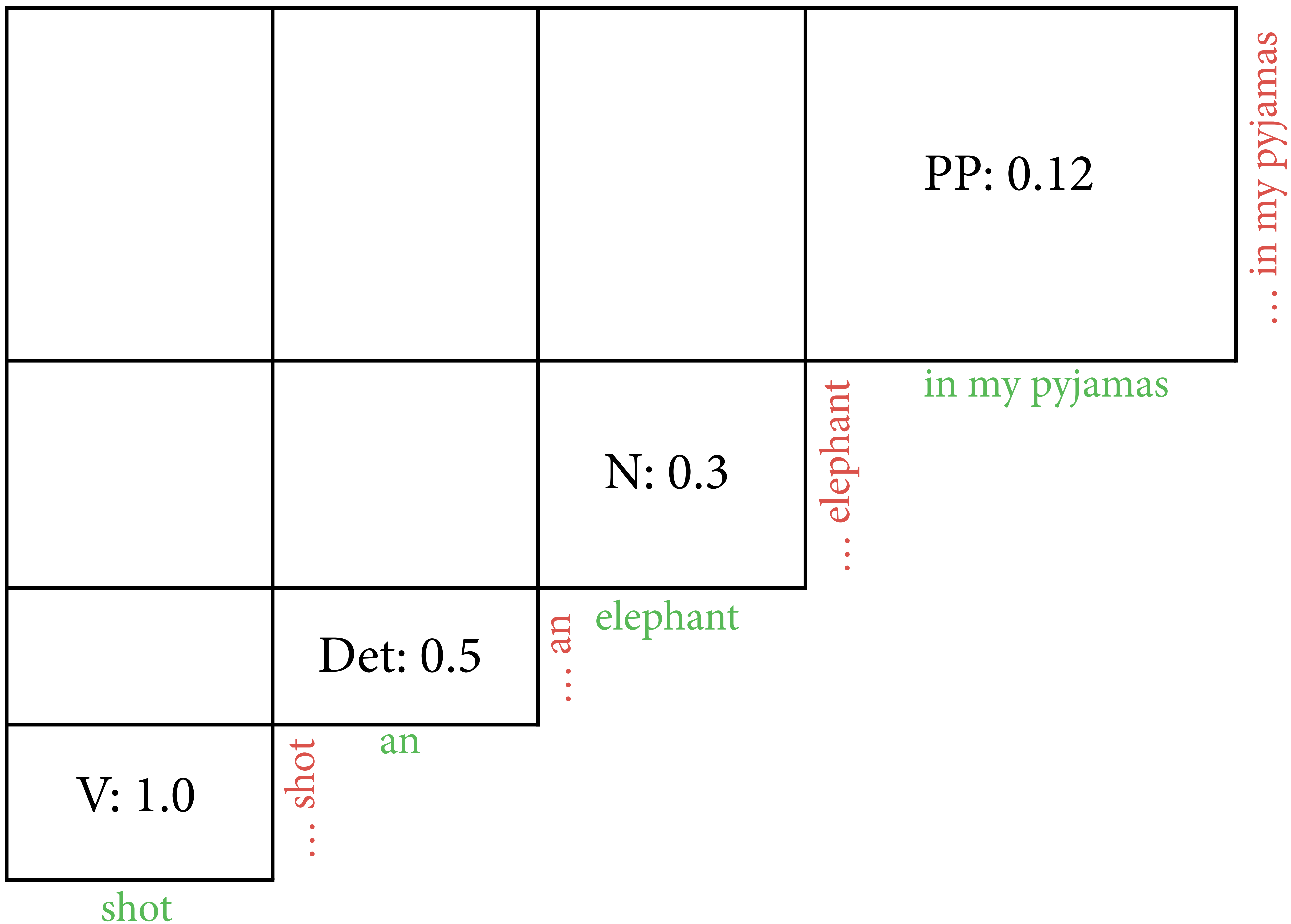
Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d:A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



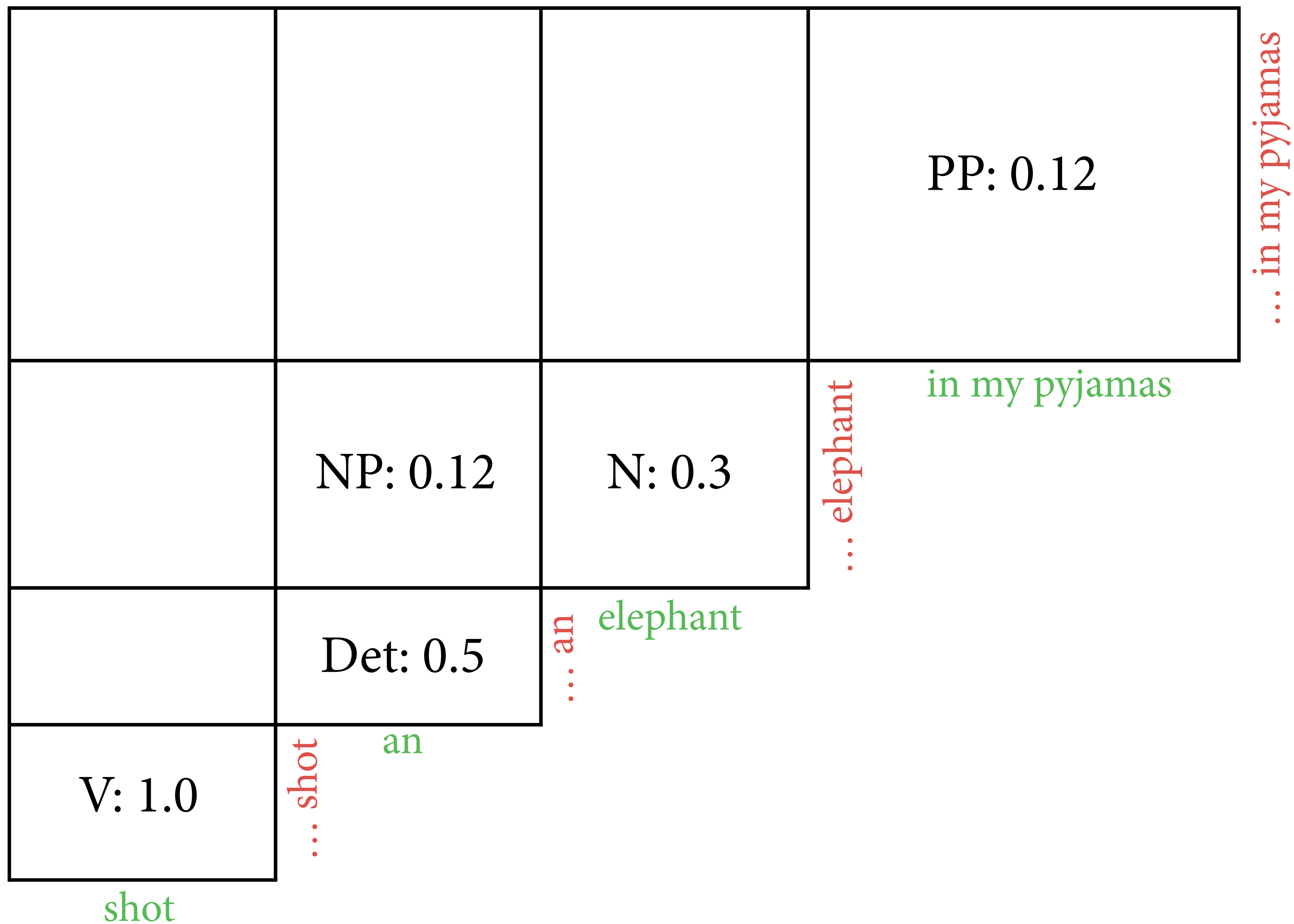
Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d: A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d:A\Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



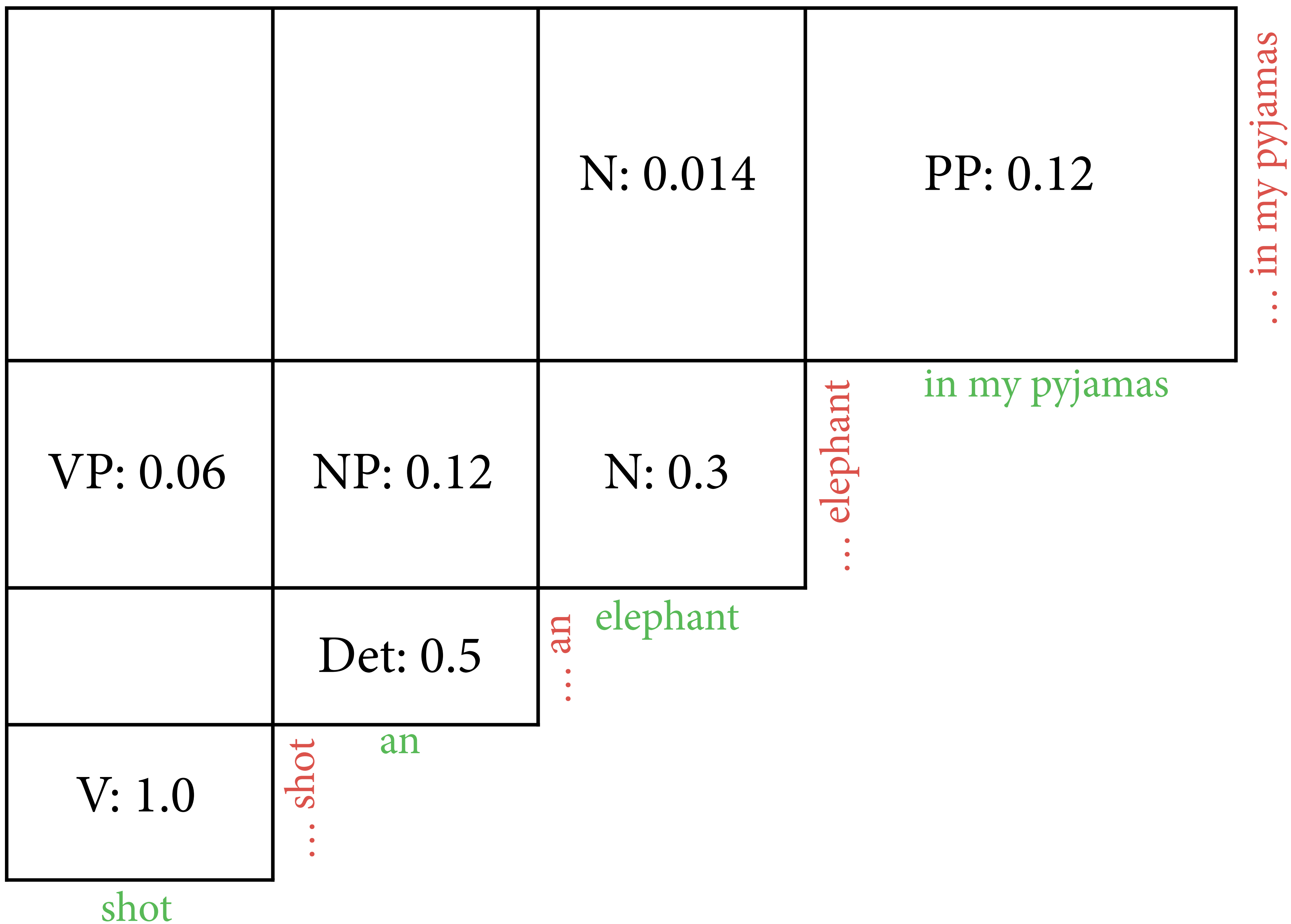
Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d: A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$

		N: 0.014	PP: 0.12	… in my pyjamas
	NP: 0.12	N: 0.3		… elephant
	Det: 0.5			… an
V: 1.0				… shot
shot	an	elephant	in my pyjamas	

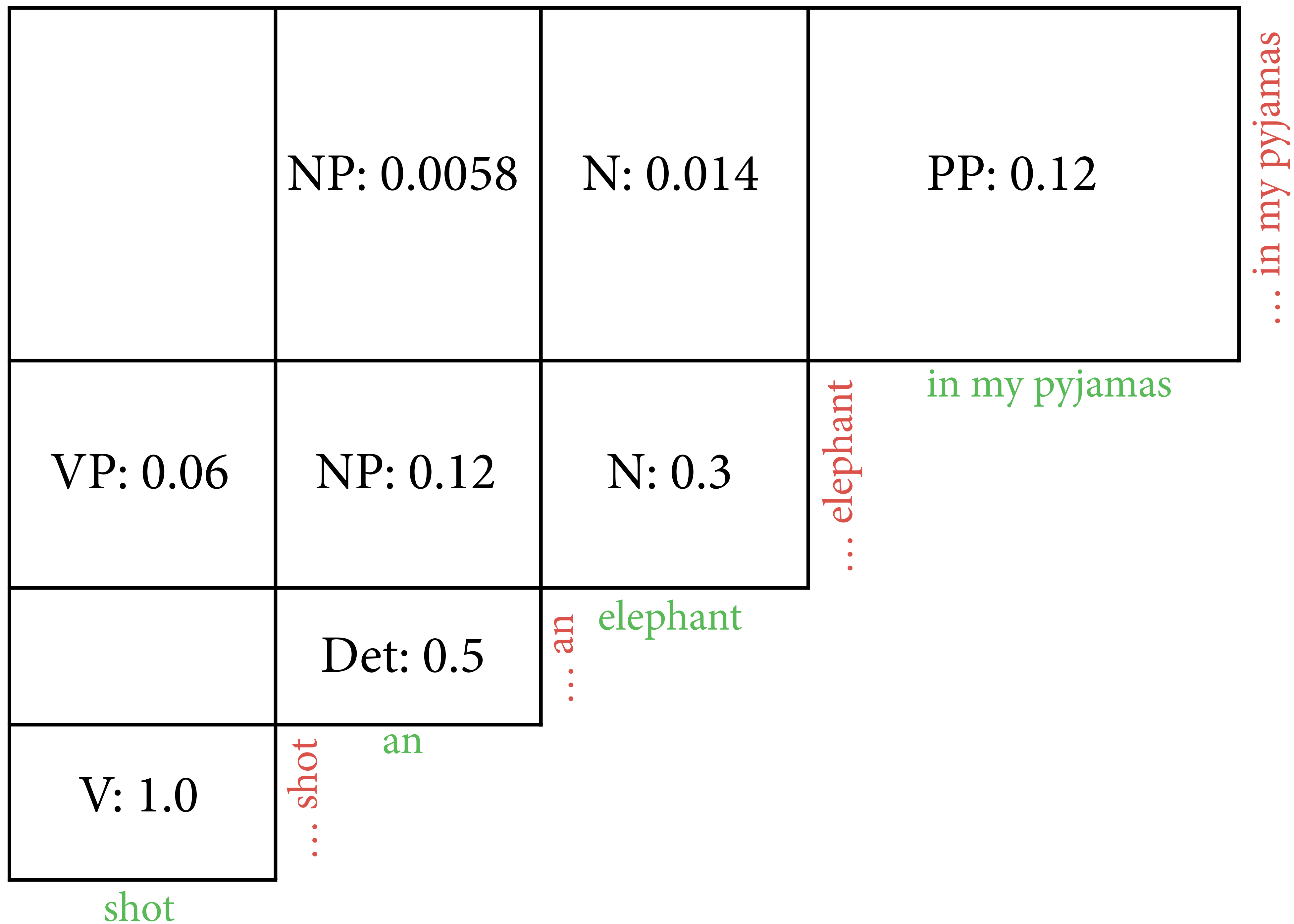
Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d: A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



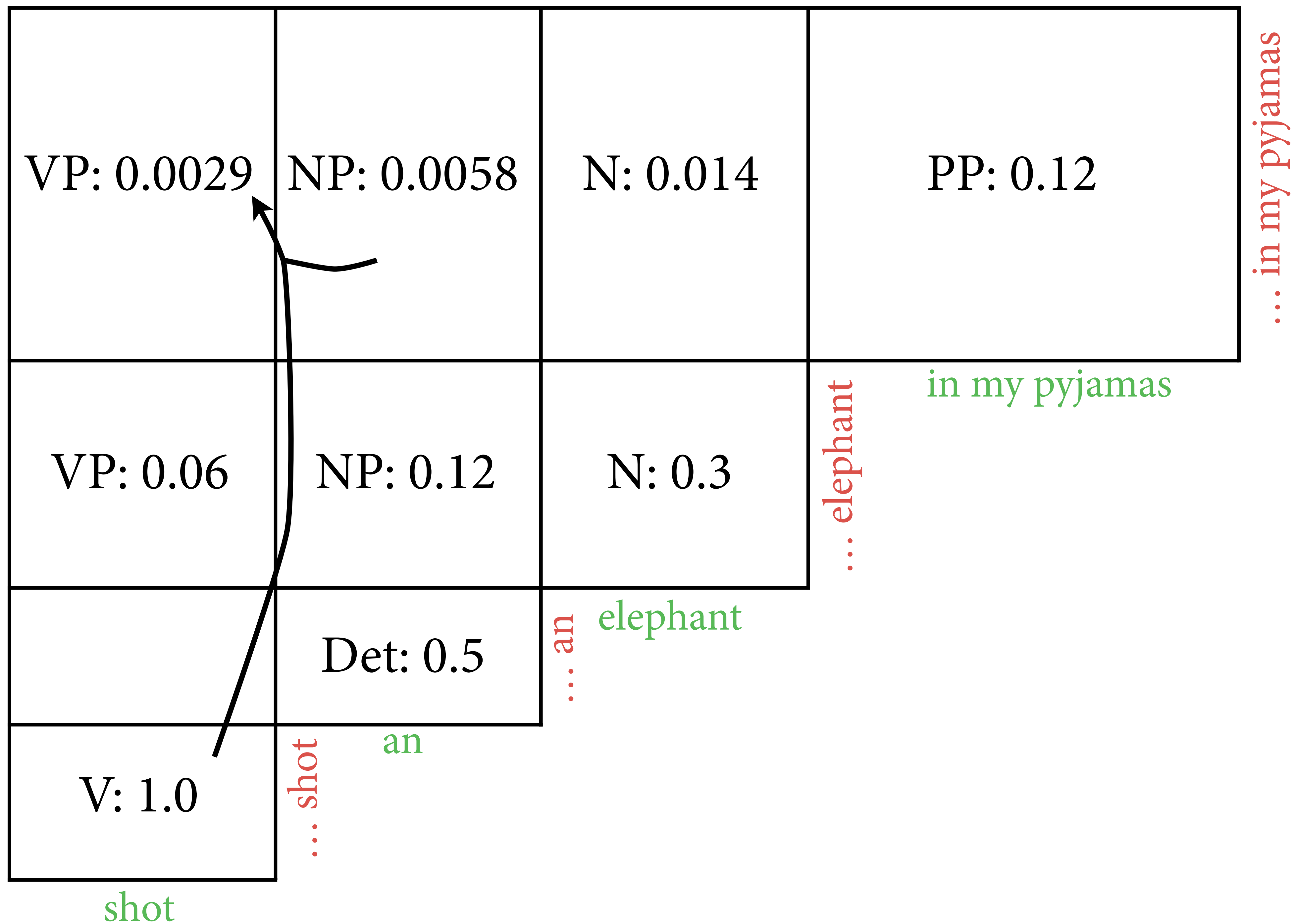
Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d:A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



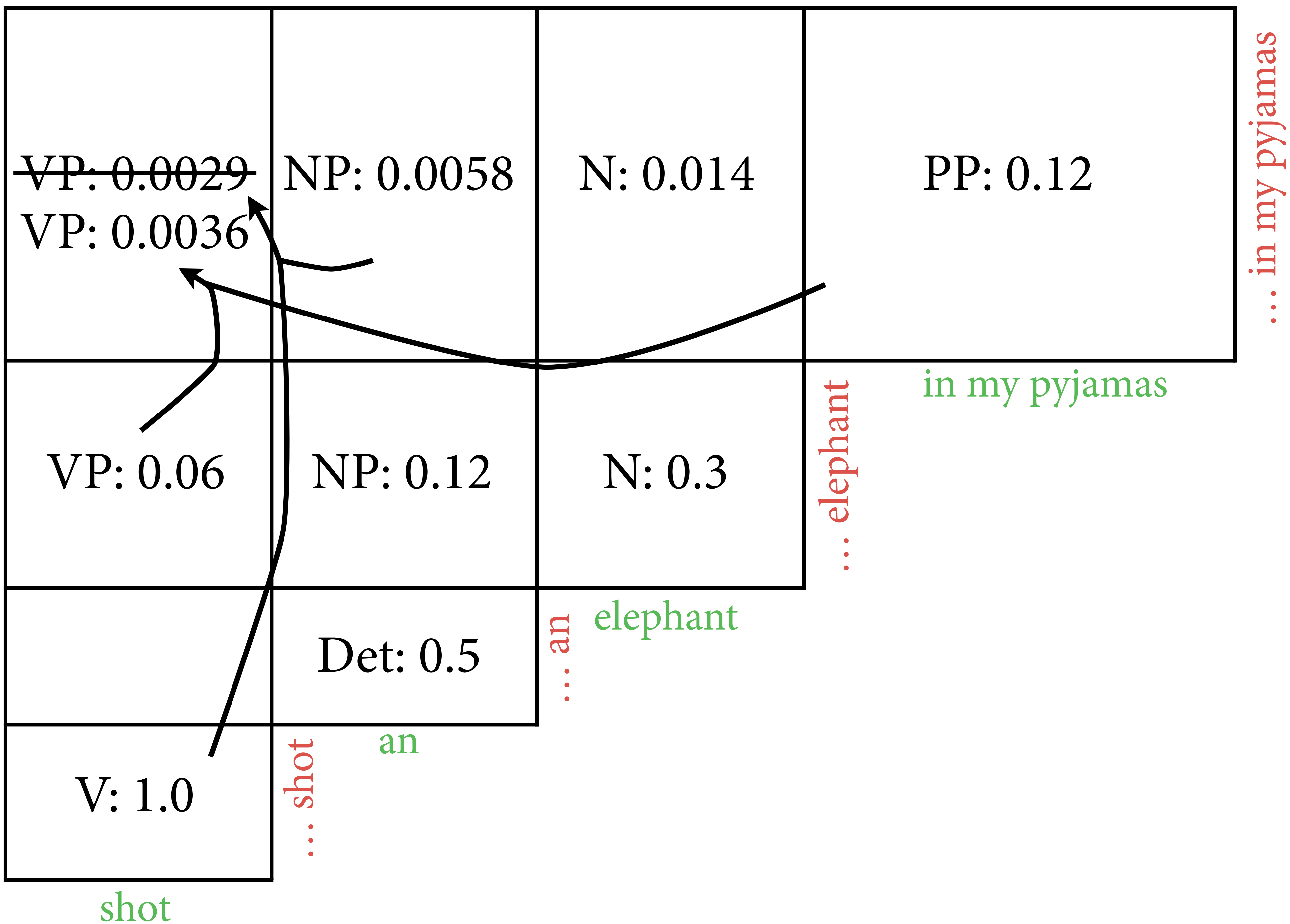
Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d:A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



Viterbi-CKY in action

Viterbi CKY parse chart: $\mathrm{Ch}(i,k) = \{(A,p) \mid p = \max_{d:A \Rightarrow^* w_i \ldots w_{k-1}} P(d)\}$



Remarks

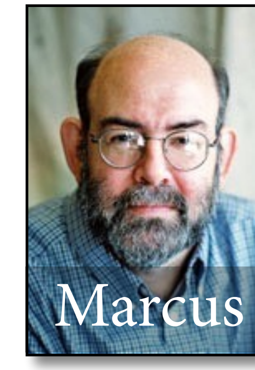
- Viterbi CKY has exactly the same nested loops as the ordinary CKY parser.
  - computing V in addition to Ch only changes constant factor
  - thus asymptotic runtime remains $O(n^3)$
- Compute optimal parse by storing backpointers.
  - same backpointers as in ordinary CKY
  - sufficient to store the *best* backpointer for each (A,i,k) if we only care about best parse (and not all parses), i.e. actually uses less memory than ordinary CKY

Obtaining the PCFG

- How to obtain the CFG?
  - write by hand
  - derive from *treebank*
  - *grammar induction* from raw text

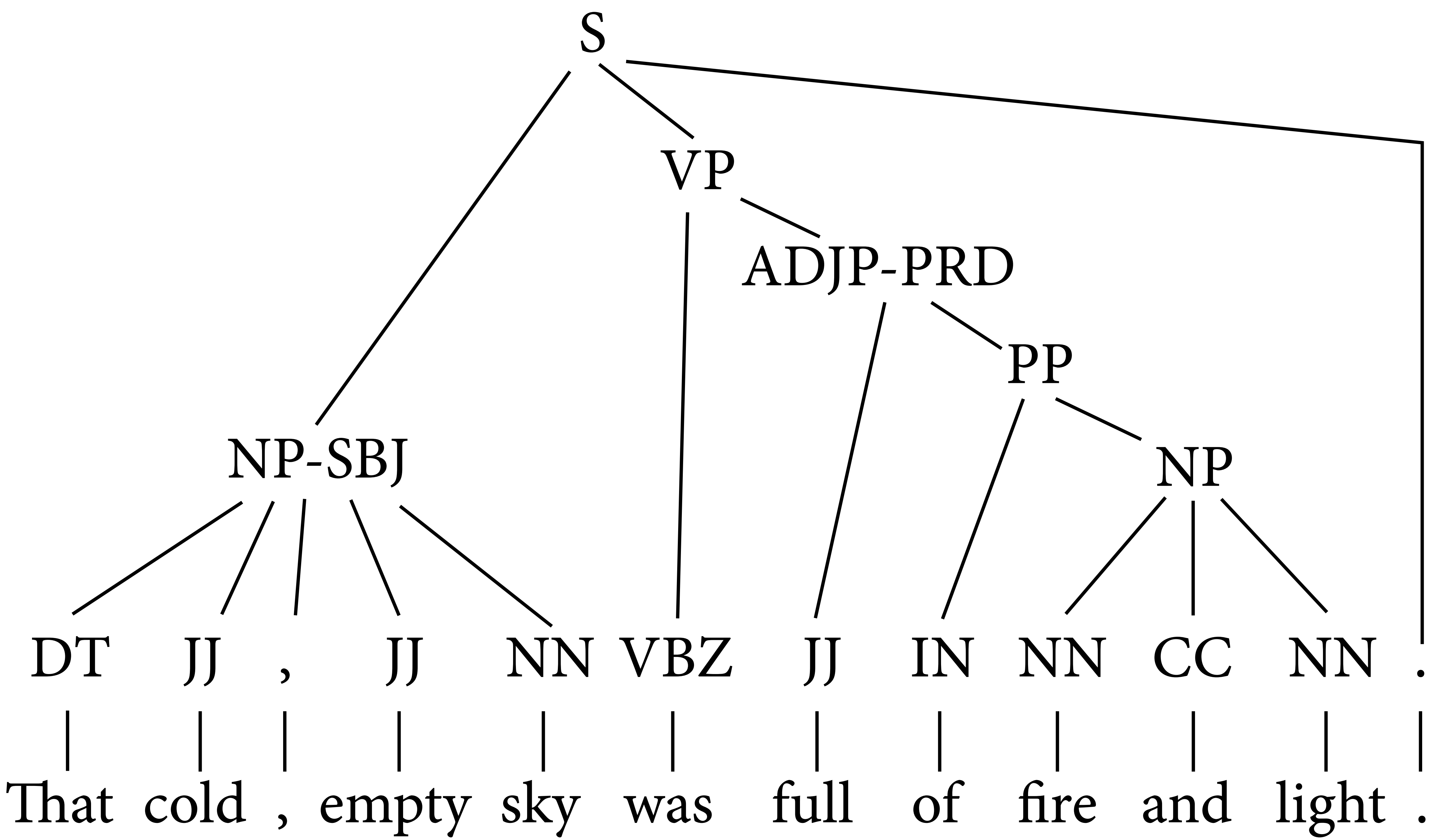
- How to obtain the rule probabilities once we have the CFG?
  - maximum likelihood estimation from treebank
  - EM training from raw text (inside-outside algorithm)

The Penn Treebank



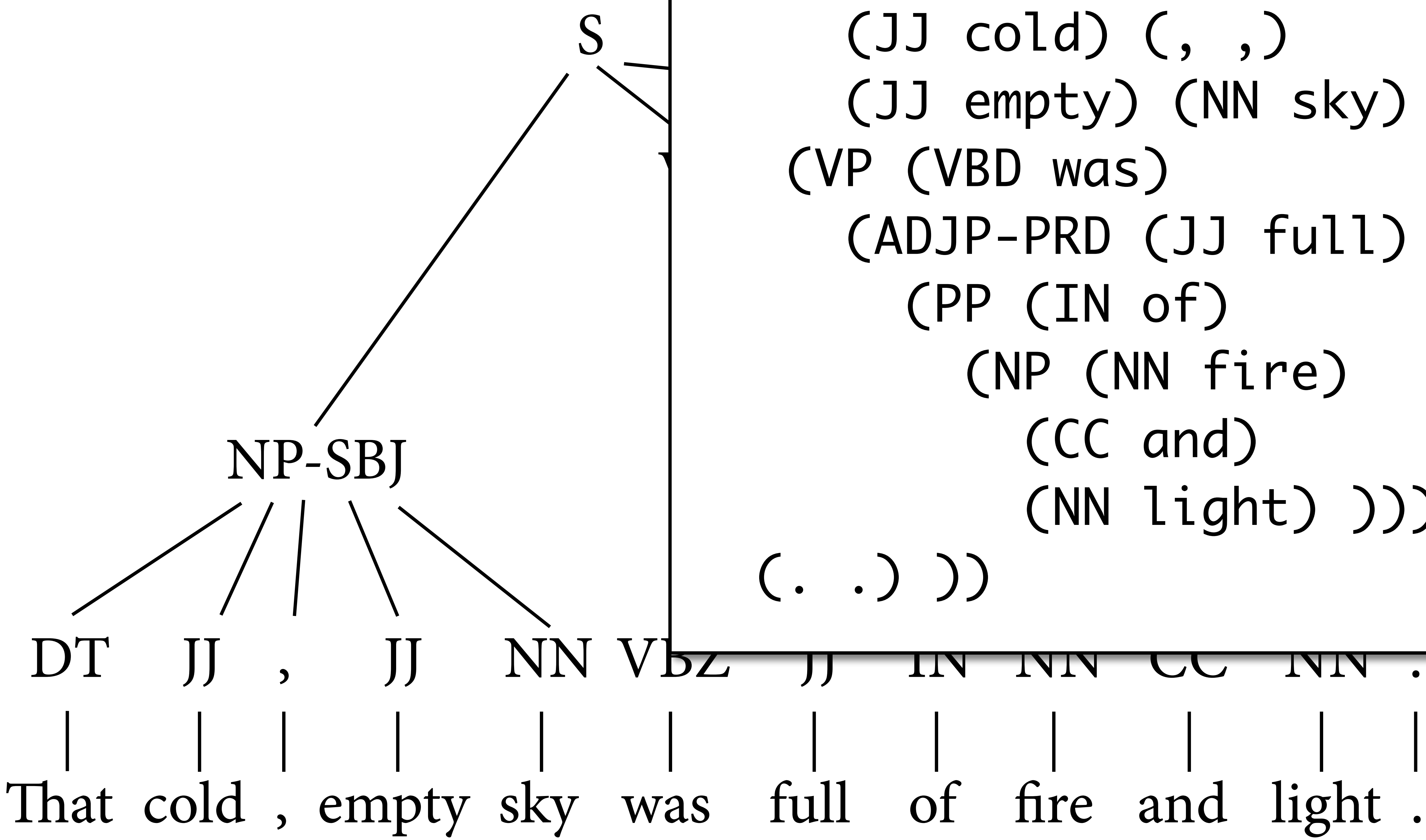
- Large (in the mid-90s) quantity of text, annotated with POS tags and syntactic structures.
- Consists of several sub-corpora:
  - Wall Street Journal: 1 year of news text, 1 million words
  - Brown corpus: balanced corpus, 1 million words
  - ATIS: dialogues on flight bookings, 5000 words
  - Switchboard: spoken dialogue, 3 million words
- WSJ PTB is standard corpus for training and evaluating PCFG parsers.

Annotation format



Annotation format

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
 (. .) ))
```



Reading off grammar

- Can directly read off "grammar in annotators' heads" from trees in treebank.

- Yields very large CFG, e.g. 4500 rules for VP:
  VP → VBD PP
  VP → VBD PP PP
  VP → VBD PP PP PP
  VP → VBD PP PP PP PP
  VP → VBD ADVP PP
  VP → VBD PP ADVP
  …
  VP → VBD PP PP PP PP PP ADVP PP

Reading off grammar

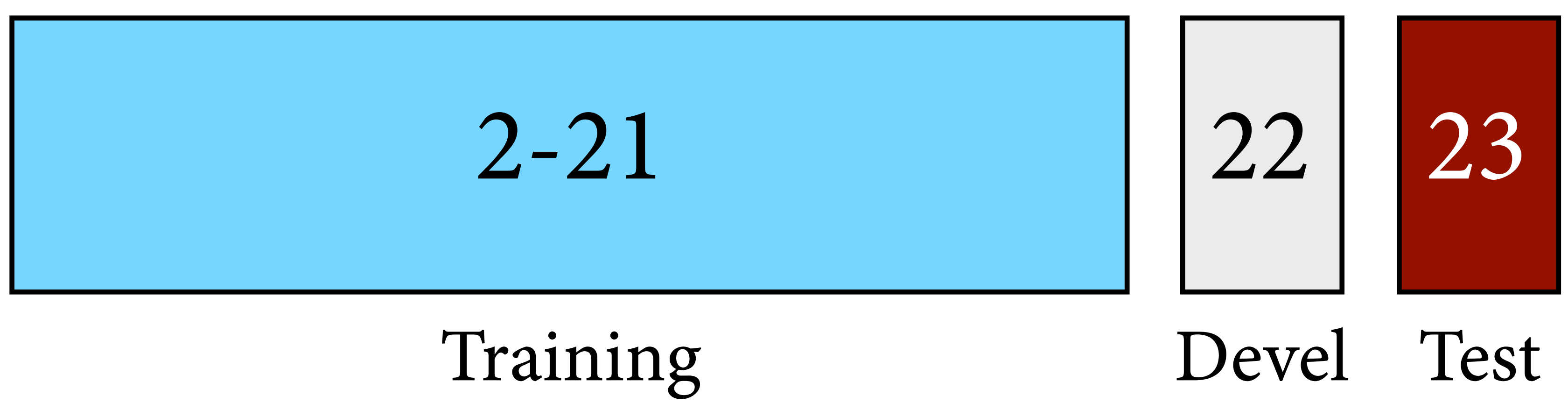
- Can directly read off "grammar in annotators' heads" from trees in treebank.

- Yields very large CFG, e.g. 4500 rules for VP:
  VP → VBD PP
  VP → VBD PP P
  VP → VBD PP P
  VP → VBD PP P
  VP → VBD ADVP PP
  VP → VBD PP ADVP
  …
  VP → VBD PP PP PP PP PP ADVP PP

"This mostly happens because we go from football in the fall to lifting in the winter to football again in the spring."

Evaluation

- Step 1: Decide on training and test corpus.
  For WSJ corpus, there is a conventional split by sections:

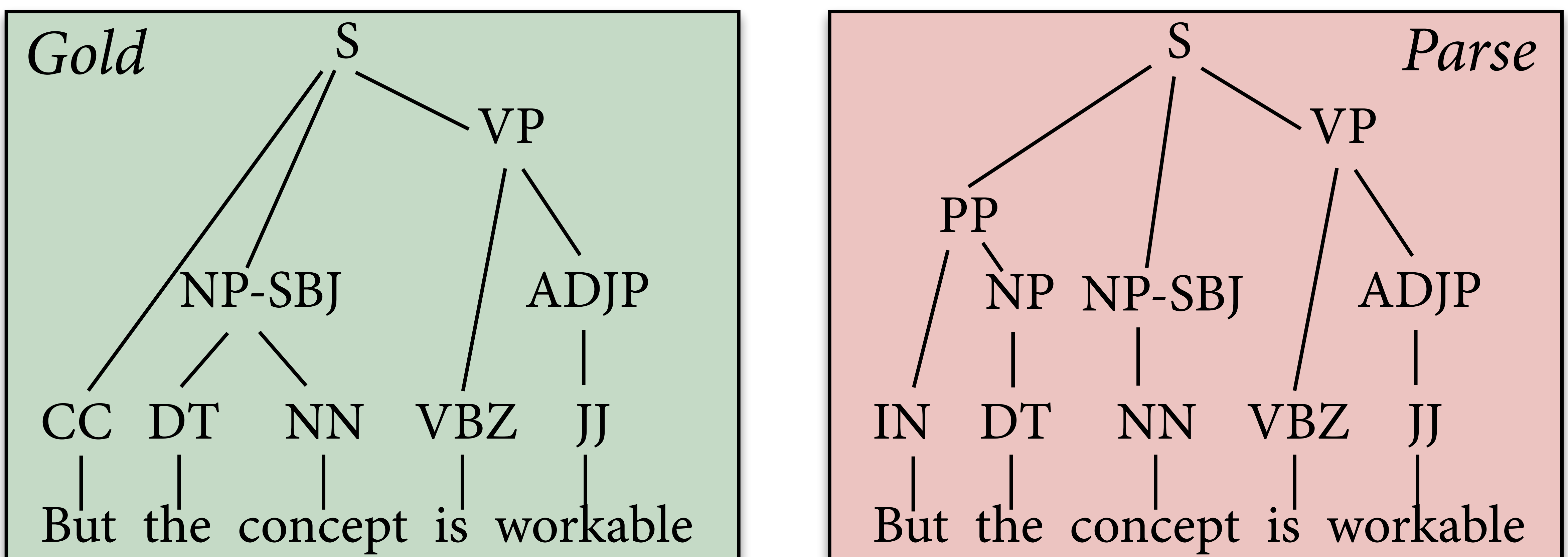


Evaluation

- Step 2: How should we measure the accuracy of the parser?
- Straightforward idea: Measure “exact match”, i.e. proportion of gold standard trees that parser got right.
- This is too strict:
    - parser makes many decisions in parsing a sentence
    - a single incorrect parsing decision makes tree “wrong”
    - want more fine-grained measure

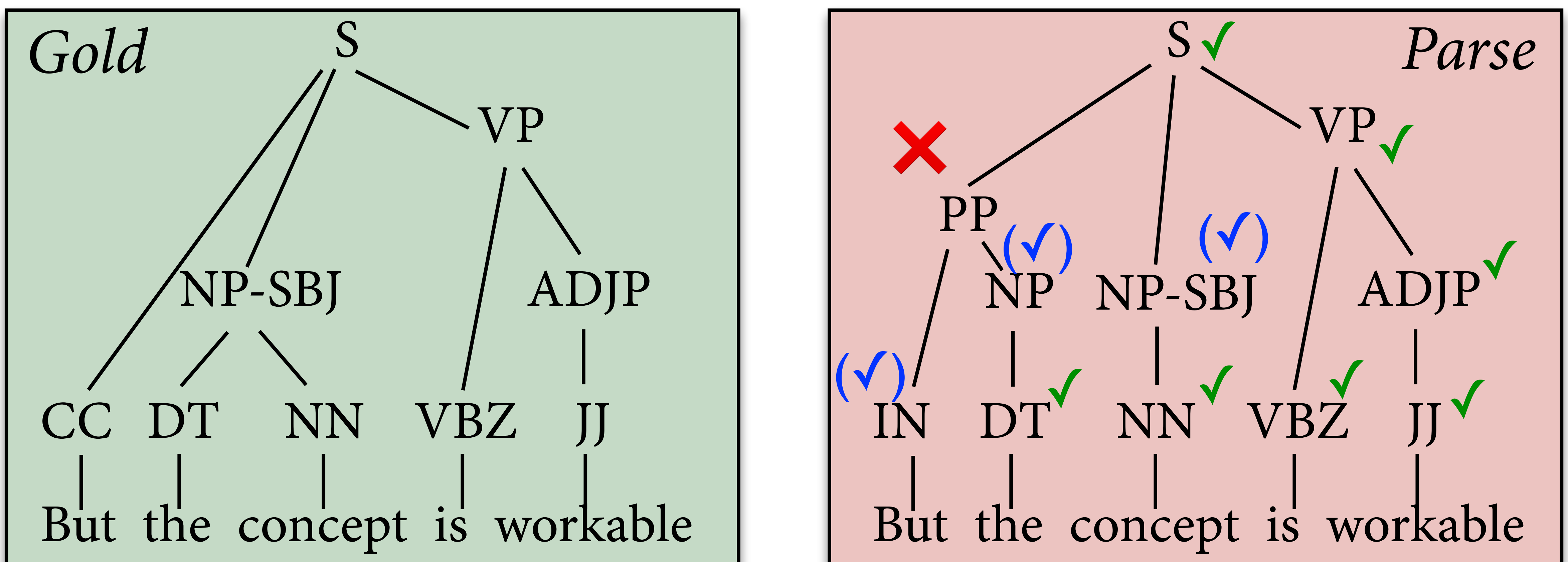
Comparing parse trees

- Idea 2 (PARSEVAL): Compare *structure* of parse tree and gold standard tree.
  - Labeled: Which *constituents* (span + syntactic category) of one tree also occur in the other?
  - Unlabeled: How do the trees bracket the *substrings* of the sentence (ignoring syntactic categories)?



Precision

What proportion of constituents in *parse tree* is also present in *gold tree?*

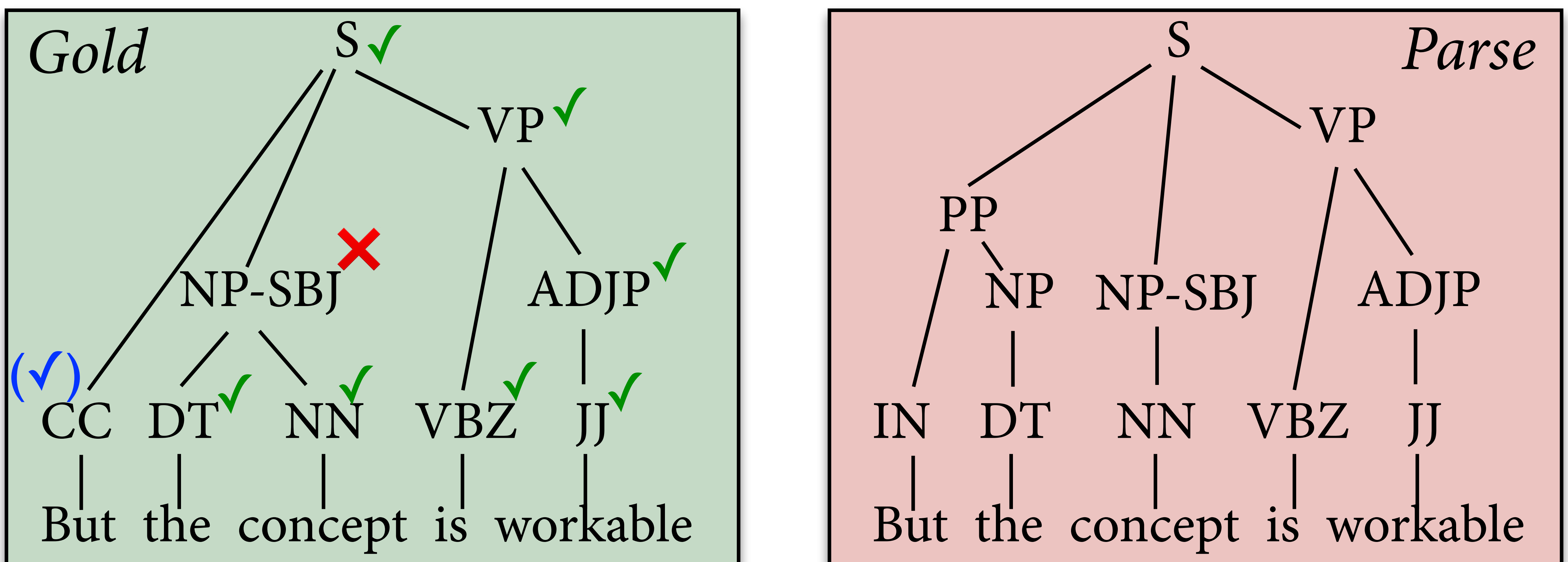


Labeled Precision = 7 / 11 = 63.6%

Unlabeled Precision = 10 / 11 = 90.9%

Recall

What proportion of constituents in *gold tree* is also present in *parse tree?*



Labeled Recall = 7 / 9 = 77.8%
Unlabeled Recall = 8 / 9 = 88.9%

F-Score

- Precision and recall measure opposing qualities of a parser ("soundness" and "completeness")

- Summarize both together in the *f-score:*

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

- In the example, we have labeled f-score 70.0 and unlabeled f-score 89.9.

Summary

- PCFGs extend CFGs with rule probabilities.
  - Events of random process are nonterminal expansion steps. These are all statistically independent.
  - Use Viterbi CKY parser to find most probable parse tree for a sentence in cubic time.
- Read grammars off treebanks.
  - next time: learn rule probabilities
- Evaluation of statistical parsers.