Computational Lexicology, Morphology and Syntax

Diana Trandabat

2022-2023

CKY Parsing

- Cocke-Kasami-Younger parsing algorithm:
  - (Relatively) efficient bottom-up parsing algorithm based on tabulating substring parses to avoid repeated work
  - Approach:
    - Use a Chomsky Normal Form grammar
    - Build an (n+1) x (n+1) matrix to store subtrees
      - Upper triangular portion
    - Incrementally build parse spanning whole input string

Chomsky Normal Form

- A Chomsky Normal Form grammar is a Context-Free Grammar in which:
  - Every rule LHS (left hand side) is a non-terminal
  - Every rule RHS (right hand side) consists of either a single terminal or two non-terminals.
  - Examples:
    - A → B C
    - NP → Nominal PP
    - A → *a*
    - Noun → *man*
  - But not:
    - NP → *the* Nominal
    - S →VP

Chomsky Normal Form

- Any CFG can be re-written in CNF, without any loss of expressiveness.
  - That is, for any CFG, there is a corresponding CNF grammar which accepts exactly the same set of strings as the original CFG.

Dynamic Programming in CKY

- Key idea:
  - For a parse spanning substring [i,j] , there exists some k such there are parses spanning [i,k] and [k,j]
    - We can construct parses for whole sentence by building up from these stored partial parses
- So,
  - To have a rule A -> B C in [i,j],
    - We must have B in [i,k] and C in [k,j], for some $i<k<j$
      - CNF grammar forces this for all $j>i+1$

CKY

- Given an input string S of length n,
  - Build table (n+1) x (n+1)
  - Indexes correspond to inter-word positions
    - W.g., 0 Book 1 That 2 Flight 3
- Cells [i,j] contain sets of non-terminals of ALL constituents spanning i,j
  - [j-1,j] contains pre-terminals
  - If [0,n] contains S, the input is recognized

Recognising strings with CKY

Example input: *The flight includes a meal.*

- The CKY algorithm proceeds by:

1. Splitting the input into words and indexing each position.

   (0) the (1) flight (2) includes (3) a (4) meal (5)

2. Setting up a table. For a sentence of length *n*, we need (n+1) rows and (n+1) columns.
3. Traversing the input sentence left-to-right
4. Use the table to store constituents and their span.

A CNF CFG for CKY

- S → NP VP
- NP → Det N
- VP → V NP
- V → includes
- Det → the
- Det → a
- N → meal
- N → flight

The table

Rule: Det → *the*

[0,1] for “the”

	1	2	3	4	5
0	Det				S
1					
2					
3					
4					

the flight includes a meal

The table

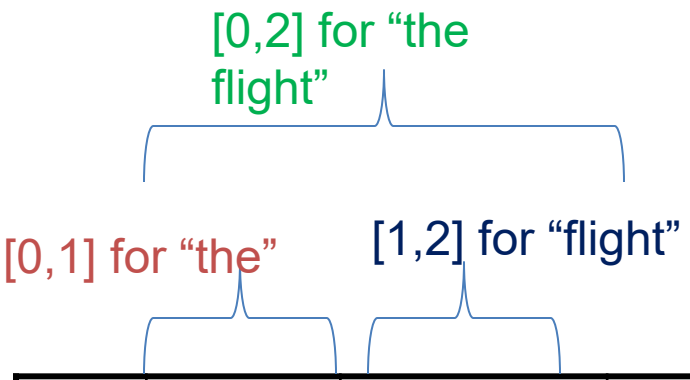
Rule1: Det → *the*
Rule 2: N → flight

[0,1] for "the" [1,2] for "flight"

	1	2	3	4	5
0	**Det**				S
1		**N**			
2					
3					
4					

the flight includes a meal

The table



Rule1: Det → *the*
Rule 2: N → flight
Rule 3: NP → Det N

	1	2	3	4	5
0	Det	NP			S
1		N			
2					
3					
4					

the flight includes a meal

CKY algorithm: two components

- **Lexical step:**

  for *j* **from** 1 **to** length(string) **do**:

  let *w* be the word in position *j*

  find all rules ending in *w* of the form X → *w*

  put X in table[j-1,j]

- **Syntactic step:**

  for i = j-2 **to** 0 **do**:

  for k = i+1 **to** j-1 **do**:

  for each rule of the form A→ B C **do**:

  if B **is in** table[i,k] **&** C is **in** table[k,j] **then**

  add A to table[i,j]

CKY: lexical step (j = 1)

- *The flight includes a meal.*

Lexical lookup

X → the

• Matches Det → the

	1	2	3	4	5
0	Det				
1					
2					
3					
4					

CKY: lexical step (j = 2)

- *The flight includes a meal.*

Lexical lookup

X → flight

• Matches N → flight

	1	2	3	4	5
0	Det				
1		N			
2					
3					
4					

CKY: syntactic step (j = 2)

- *The flight includes a meal.*

Syntactic lookup:

• look backwards and see if there is any rule that will cover what we've done so far.

X → Det N

•Matches NP → Det N

	1	2	3	4	5
0	Det	NP			
1		N			
2					
3					
4					

CKY: lexical step (j = 3)

- *The flight includes a meal.*

Lexical lookup
• Matches V → includes

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3					
4					

CKY: lexical step (j = 3)

- *The flight includes a meal.*

Syntactic lookup

• There are no rules in our grammar that will cover N V

X → N V is not found

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3					
4					

CKY: lexical step (j = 4)

- *The flight includes a meal.*

Lexical lookup
• Matches Det → a

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	
4					

CKY: lexical step (j = 5)

- *The flight includes a meal.*

Lexical lookup
• Matches N → meal

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	
4					N

CKY: syntactic step (j = 5)

- *The flight includes a meal.*

Syntactic lookup
• We find that we have NP
→ Det N

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	NP
4					N

CKY: syntactic step (j = 5)

- *The flight includes a meal.*

Syntactic lookup
• We find that we have
VP → V NP

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		VP
3				Det	NP
4					N

CKY: syntactic step (j = 5)

- *The flight includes a meal.*

Syntactic lookup
• We find that we have
S → NP VP

	1	2	3	4	5
0	Det	NP			S
1		N			
2			V		VP
3				Det	NP
4					N

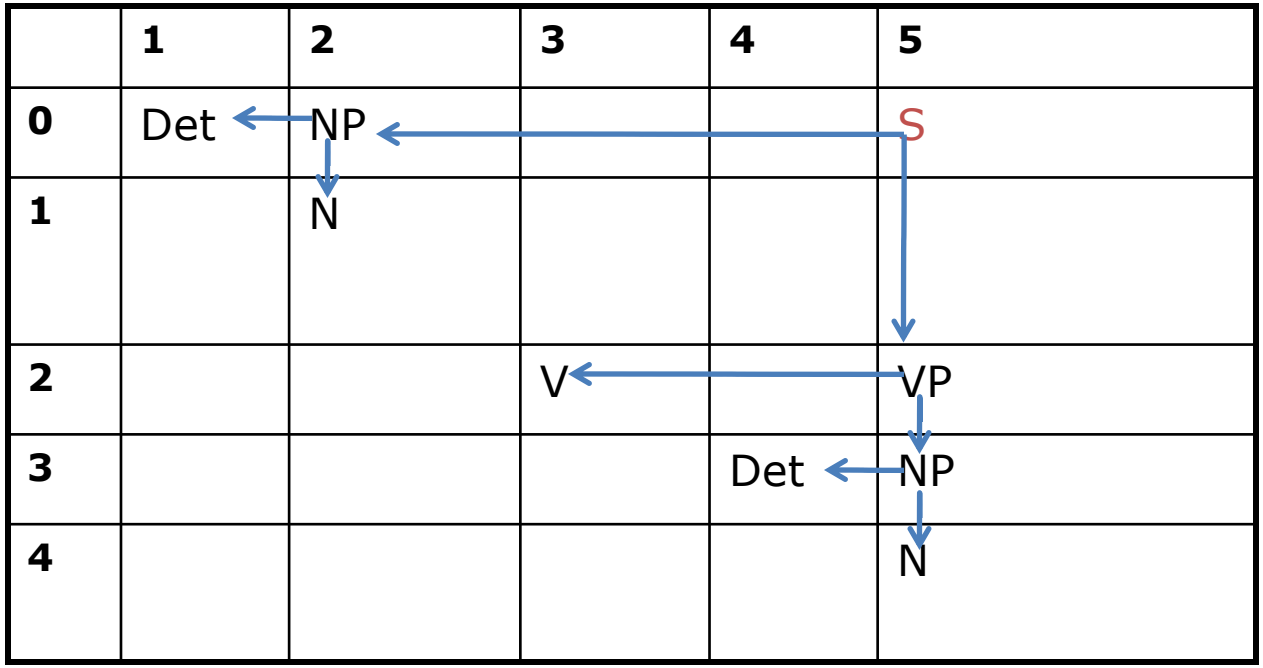
From recognition to parsing

- The procedure so far will recognise a string as a legal sentence in English.

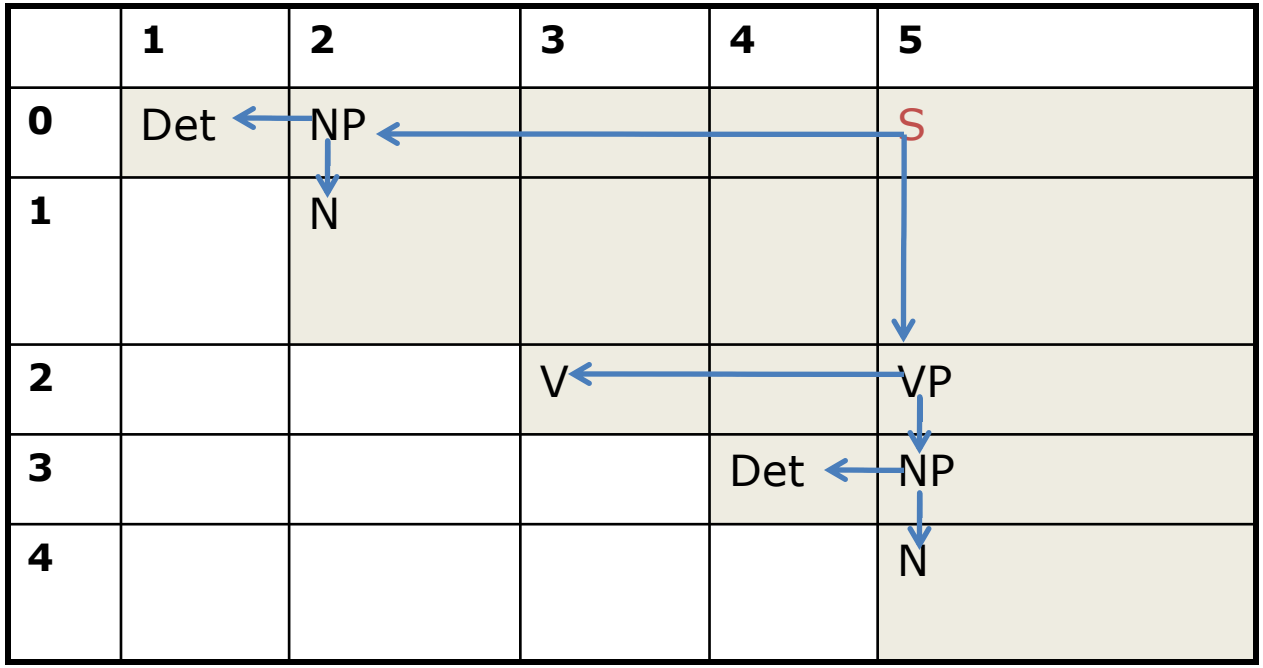
- But we'd like to get a parse tree back!

- Solution:
  - We can work our way back through the table and collect all the partial solutions into one parse tree.
  - Cells will need to be augmented with "backpointers", i.e. with a pointer to the cells that the current cell covers.

From recognition to parsing



From recognition to parsing



NB: This algorithm always fills the top “triangle” of the table!

What about ambiguity?

- The algorithm does not assume that there is only one parse tree for a sentence.
  - (Our simple grammar did not admit of any ambiguity, but this isn't realistic of course).

- There is nothing to stop it returning several parse trees.

- If there are multiple local solutions, then more than one non-terminal will be stored in a cell of the table.

Exercise

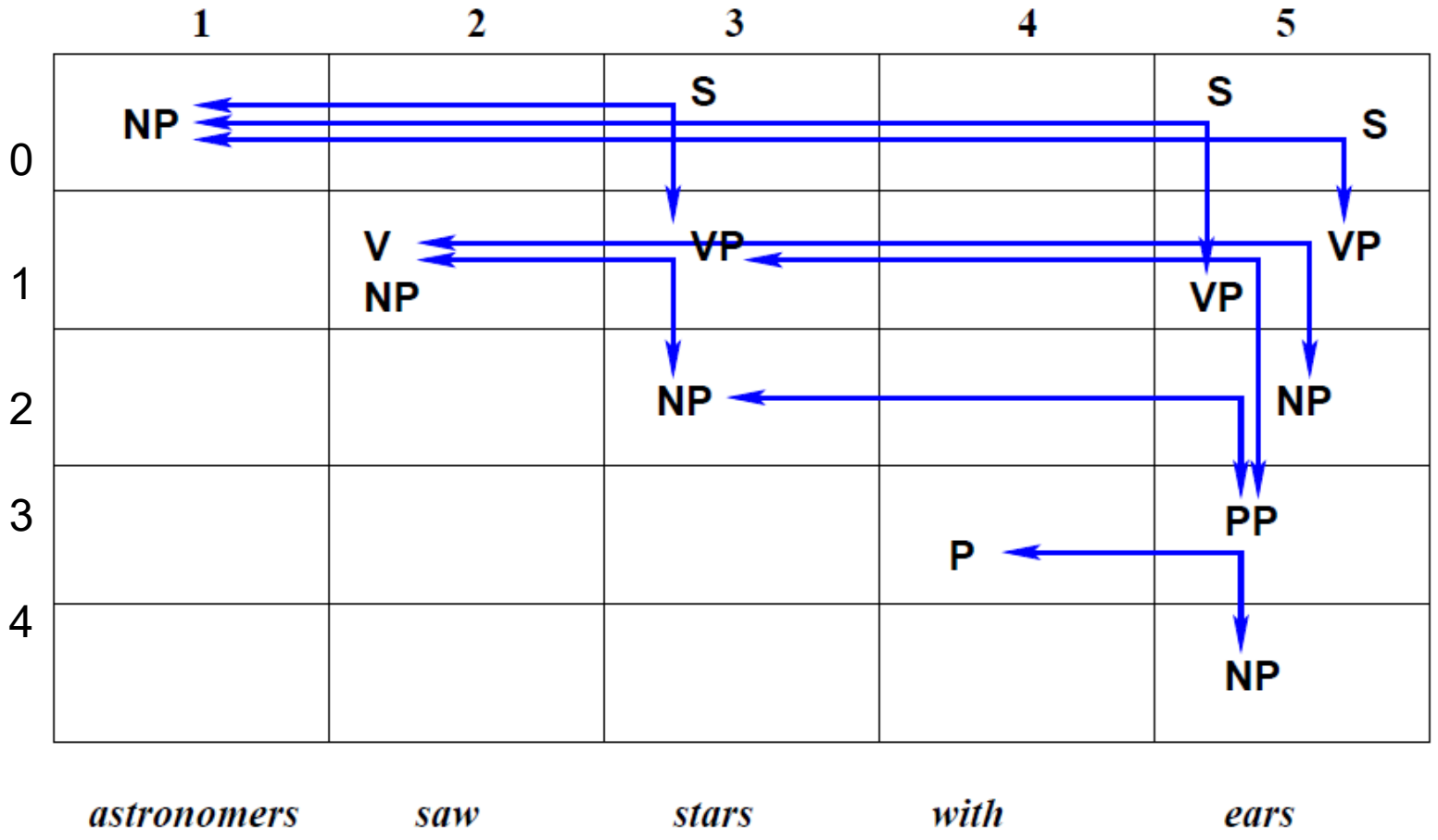
- Apply the CKY algrithm to the fllowing sentence:

  Astronomers saw stars with ears.

given the following grammar:

S - > NP VP	1.0	NP-> NP PP	0.4
PP -> P NP	1.0	NP-> astronomers	0.2
VP -> V NP	0.7	NP-> ears	0.18
VP - > VP PP	0.3	NP-> saw	0.04
P -> with	1.0	NP -> stars	0.18
V-> saw	1.0		

Exercise


1
2
3
4
5
0
1
2
3
4
NP
S
S
S
V
NP
VP
VP
VP
NP
NP
PP
P
NP
astronomers
saw
stars
with
ears

Exercise

- Now run the CKY algorithm considering also the probabilities of the rules.
- Probabilities for the first diagonal is the probability of the rule.
  - Ex. P(cell[0,1])=P(NP->astronomers)=0.2
- The probability of a cell [i, j] from the second diagonal onwards is:

P(cell[i,j]) = P(rule leading to fill the cell) * P(cell[i, j-1]) * P(cell[j+1, i])

  - Ex. P(cell[1,3])=P(VP->V NP)*P(cell[1,2])*P(cell[2,3])

$$= 0.7 * 1 * 0.18 = 0.126$$

- We will thus find the most probable parse tree.

CKY using unary rules

- The CKY algorithm can be run also with unary rules of the form A -> B
- For the first diagonal, the unary rules are applied after the lexical rule.
- For the other diagonals, for each cell, after applying the corresponding rule from the grammar (similar to the previous version of the algorithm), we need to also apply unary rules to create all new possible nonterminals.

Example of grammar using unary rules

- S → NP VP 1.0
- NP → DT NN 0.5
- NP → NNS 0.3
- NP → NP PP 0.2
- PP → P NP 1.0
- VP → VP PP 0.6
- VP → VBD NP 0.4

- DT → the 1.0
- NN → gunman 0.5
- NN → building 0.5
- VBD → sprayed 1.0
- NNS → bullets 1.0
- P → with 1.0

Exercise with unary rules

	The (1)	gunman (2)	sprayed (3)	the (4)	building (5)	with (6)	bullets (7)
0	DT						
1		NN					
2			VBD				
3				DT			
4					NN		
5						P	
6							NNS NP

- For each cell, after filling it, a rule in the grammar of the form X -> Y is searched for, where Y is the nonterminal in the cell. If such a rule exists, the new nonterminal is also added to the cell (see cell [6,7]), with backpointer to X.

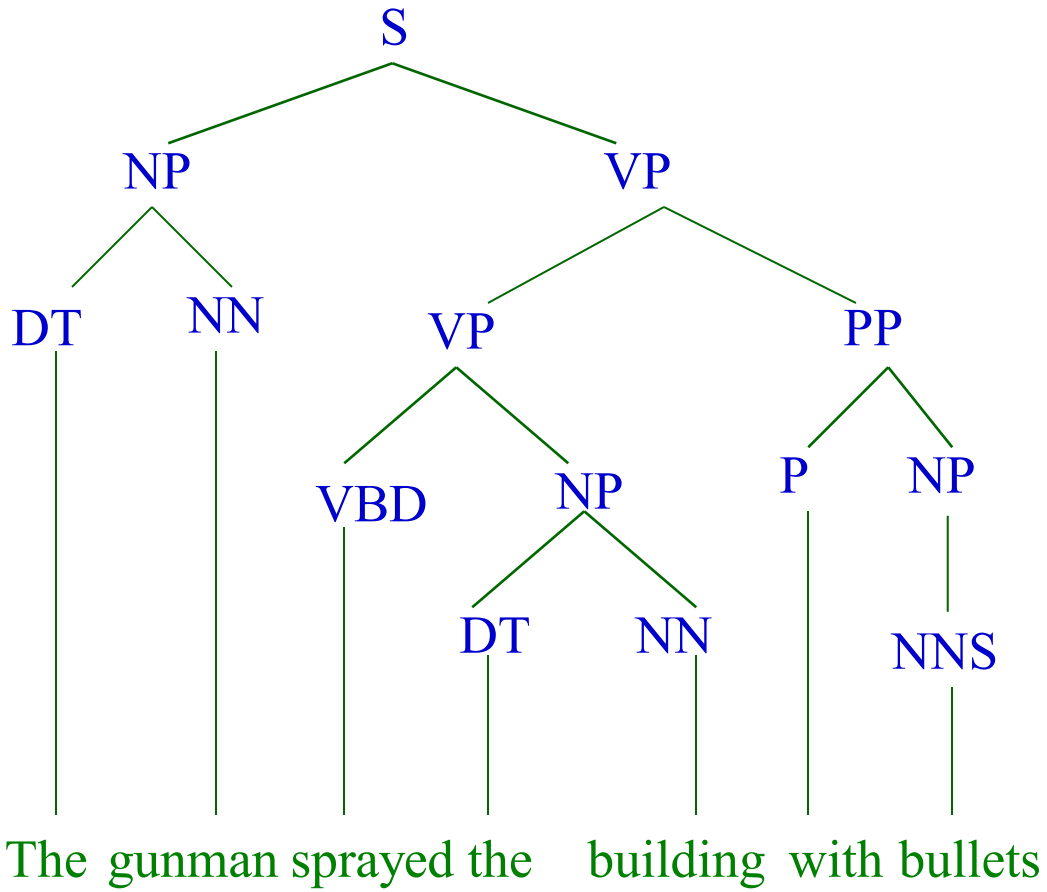
Exercise with unary rules

	The (1)	gunman (2)	sprayed (3)	the (4)	building (5)	with (6)	bullets (7)
0	DT						
1		NN					
2			VBD				
3				DT			
4					NN		
5						P	
6							NNS NP

- Continue filling the table, taking care to also apply unary rules, if any, until you obtain the two parse trees!
- You can also compute probabilities on each cell

Parse t_1`

- The gunman sprayed the building with bullets.

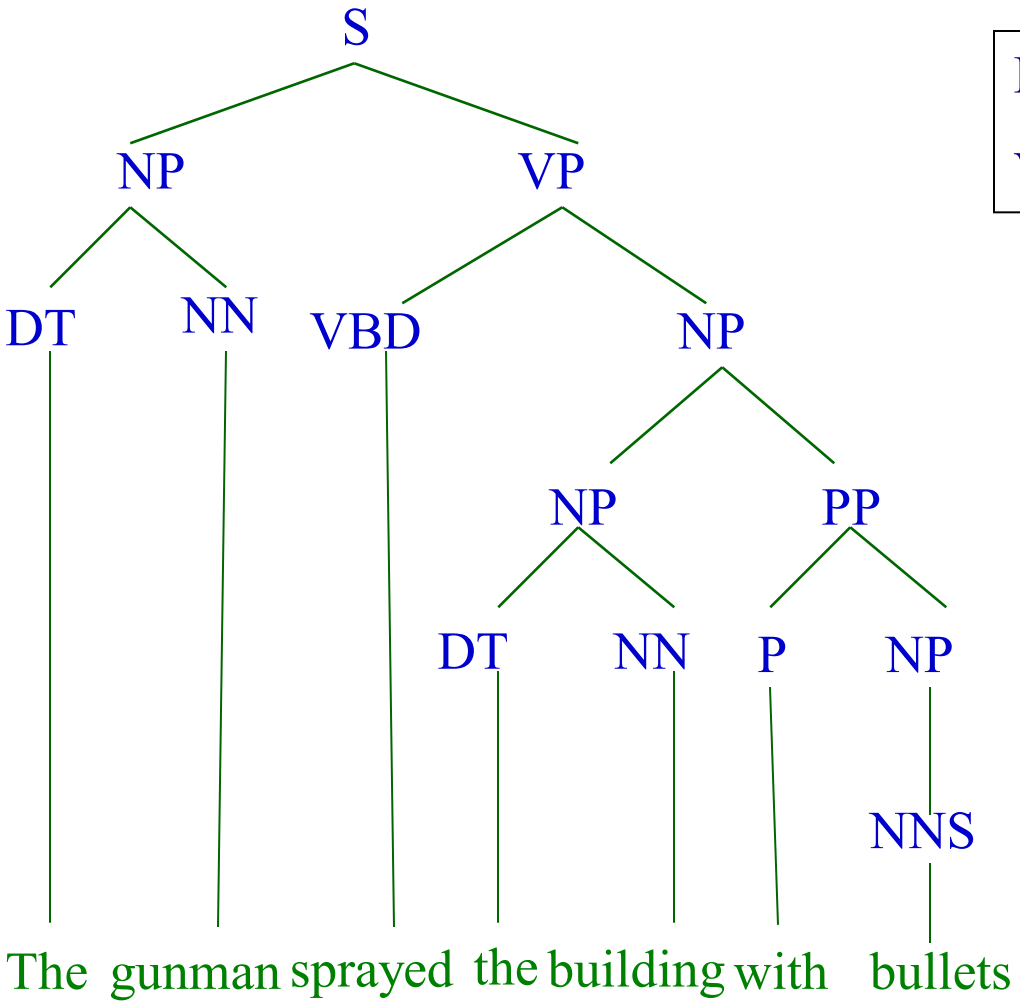


Rule used here is

$VP \rightarrow VP\ PP$

Another Parse t_2

- The gunman sprayed the building with bullets.



Rule used here is

VP → VBD NP

CKY Discussions

- Running time:
  - $O(n^3)$ where ***n*** is the length of the input string
  - Inner loop grows as square of # of non-terminals

- Expressiveness:
  - As implemented, requires CNF
    - Weakly equivalent to original grammar
    - Doesn't capture full original structure
      - Back-conversion?
        - » Can do binarization, terminal conversion
        - » Unit non-terminals require change in CKY

Great!

See you next time!