Introduction to NLP

242.

Introduction to Parsing

Parsing programming languages

```
#include <stdio.h>

int main()
{
  int n, reverse = 0;

  printf("Enter a number to reverse\n");
  scanf("%d",&n);

  while (n != 0)
  {
    reverse = reverse * 10;
    reverse = reverse + n%10;
    n = n/10;
  }
  printf("Reverse of entered number is = %d\n", reverse);

  return 0;
}
```

Parsing human languages

- Rather different than computer languages
  - Can you think in which ways?

Parsing human languages

- Rather different than computer languages
  - No types for words
  - No brackets around phrases
  - Ambiguity
    - Words
    - Parses
  - Implied information

The parsing problem

- Parsing means associating tree structures to a sentence, given a grammar (e.g., CFG)
  - There may be exactly one such tree structure
  - There may be many such structures
  - There may be none
- Grammars (e.g., CFG) are declarative
  - They don't specify how the parse tree will be constructed

Syntactic Issues

- PP attachment
  - I saw the man with the telescope
- Gaps
  - Mary likes Physics but hates Chemistry
- Coordination scope
  - Small boys and girls are playing
- Particles vs. prepositions
  - She ran up a large bill
- Gerund vs. adjective
  - Frightening kids can cause trouble

Applications of parsing

- Grammar checking
  - I want to return this shoes.
- Question answering
  - How many people in sales make $40K or more per year?
- Machine translation
  - E.g., word order – SVO vs. SOV
- Information extraction
  - *Breaking Bad* takes place in New Mexico.
- Speech generation
- Speech understanding

Introduction to NLP

247.

Parsing Evaluation

Parsing Evaluation

- Parseval: precision and recall
  - get the proper constituents
- Labeled precision and recall
  - also get the correct non-terminal labels
- F1
  - harmonic mean of precision and recall
- Crossing brackets
  - (A (B C)) vs ((A B) C)
- PTB corpus
  - training 02-21, development 22, test 23

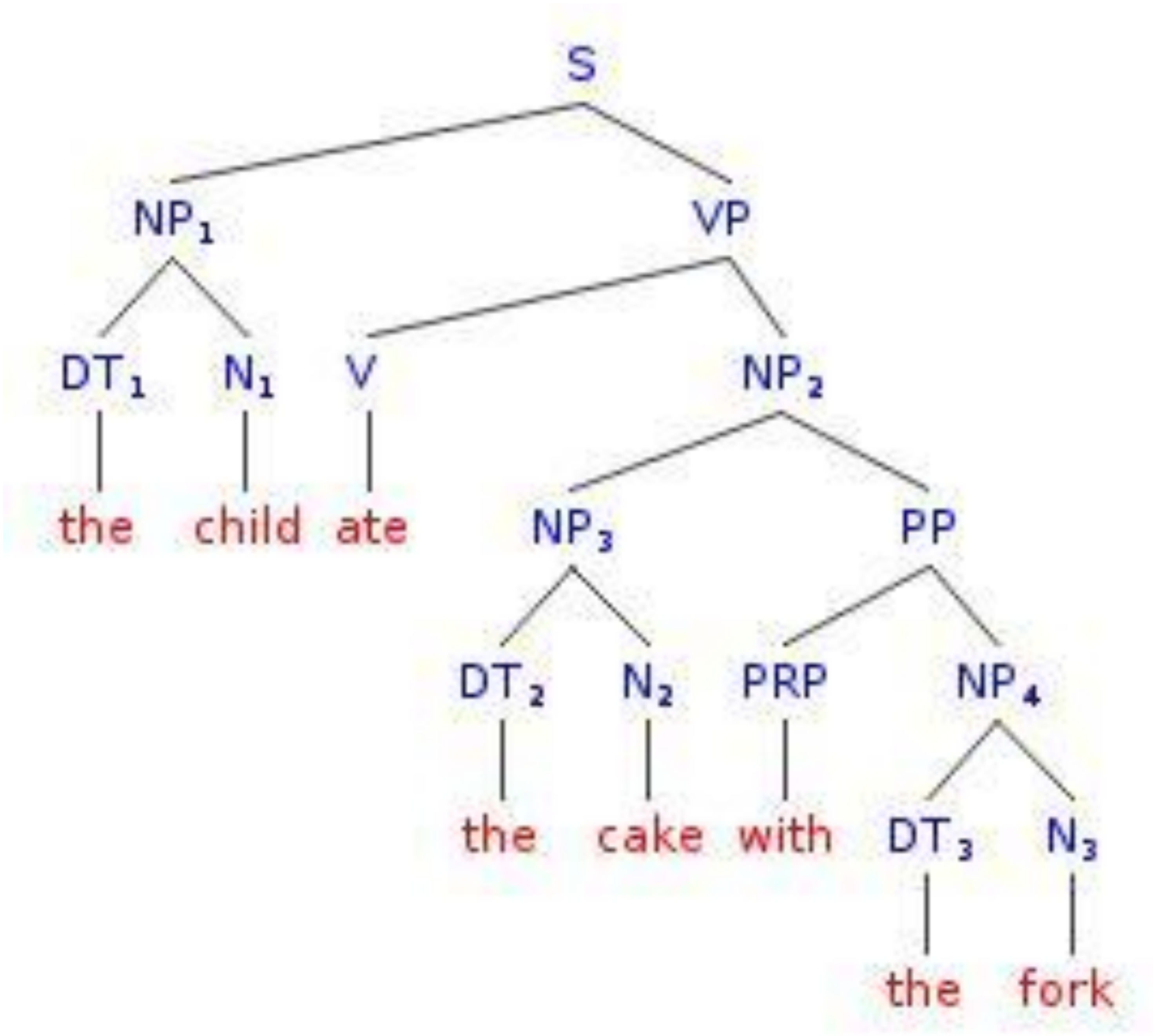
Evaluation Example

```
GOLD = (S (NP (DT The) (JJ Japanese) (JJ industrial) (NNS companies))
          (VP (MD should) (VP (VB know) (ADVP (JJR better)))) (. .)

CHAR = (S (NP (DT The) (JJ Japanese) (JJ industrial) (NNS companies))
          (VP (MD should) (VP (VB know)) ((ADVP (RBR better)))) (. .))
```

Introduction to NLP

243.

Classic parsing methods




```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

Parsing as search

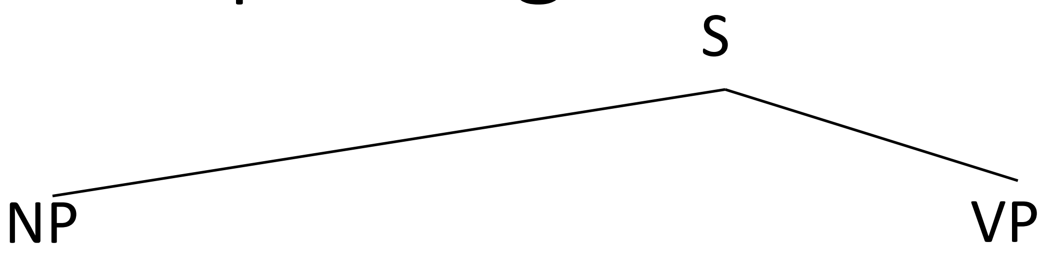
- There are two types of constraints on the parses
  - From the input sentence
  - From the grammar
- Therefore, two general approaches to parsing
  - Top-down
  - Bottom-up

Top down parsing

S

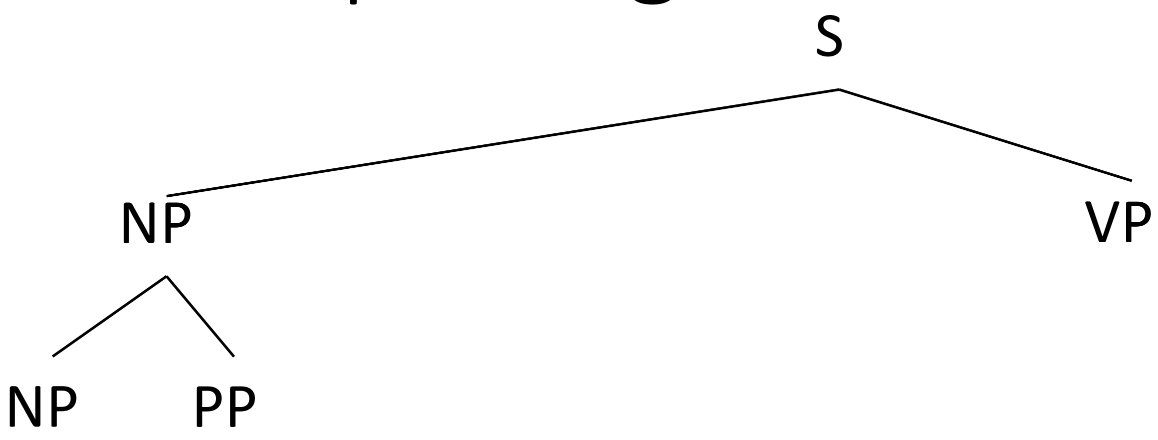
```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

Top down parsing



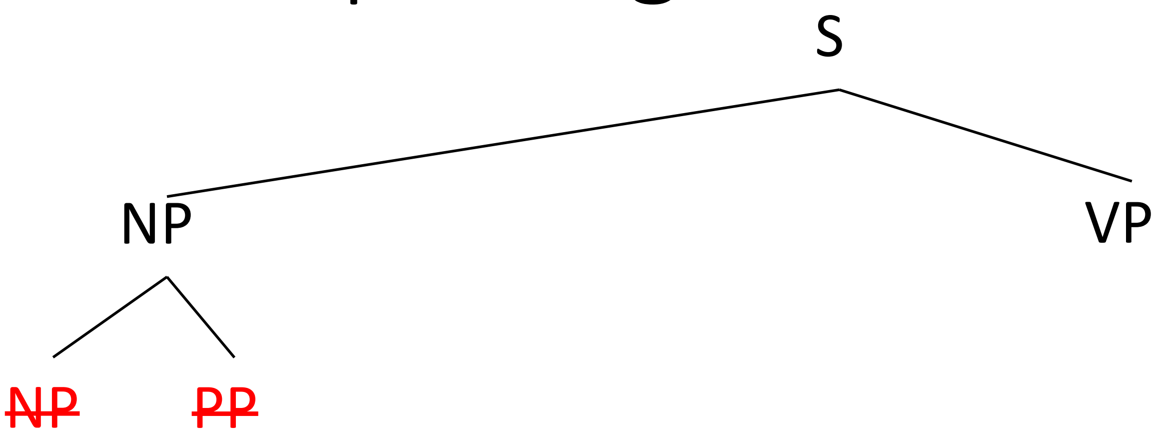

```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

Top down parsing



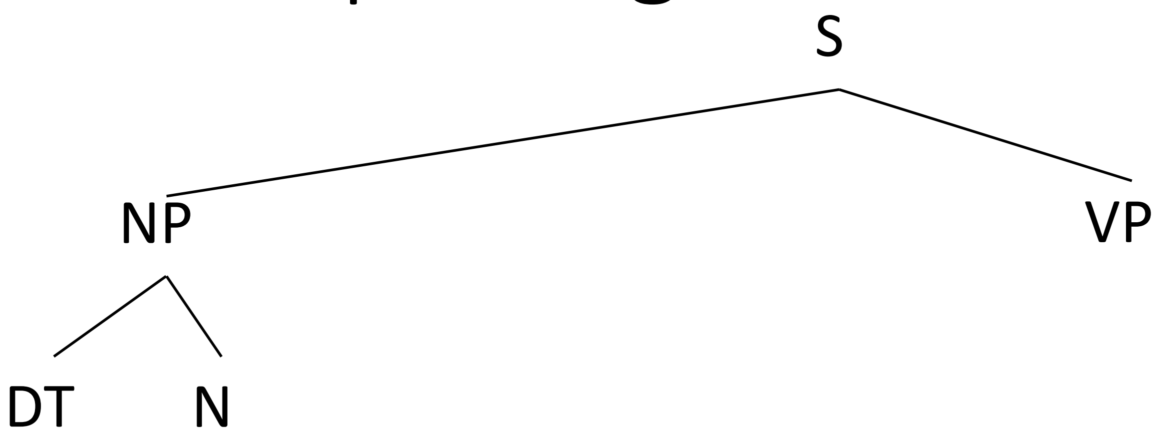

```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

Top down parsing




```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

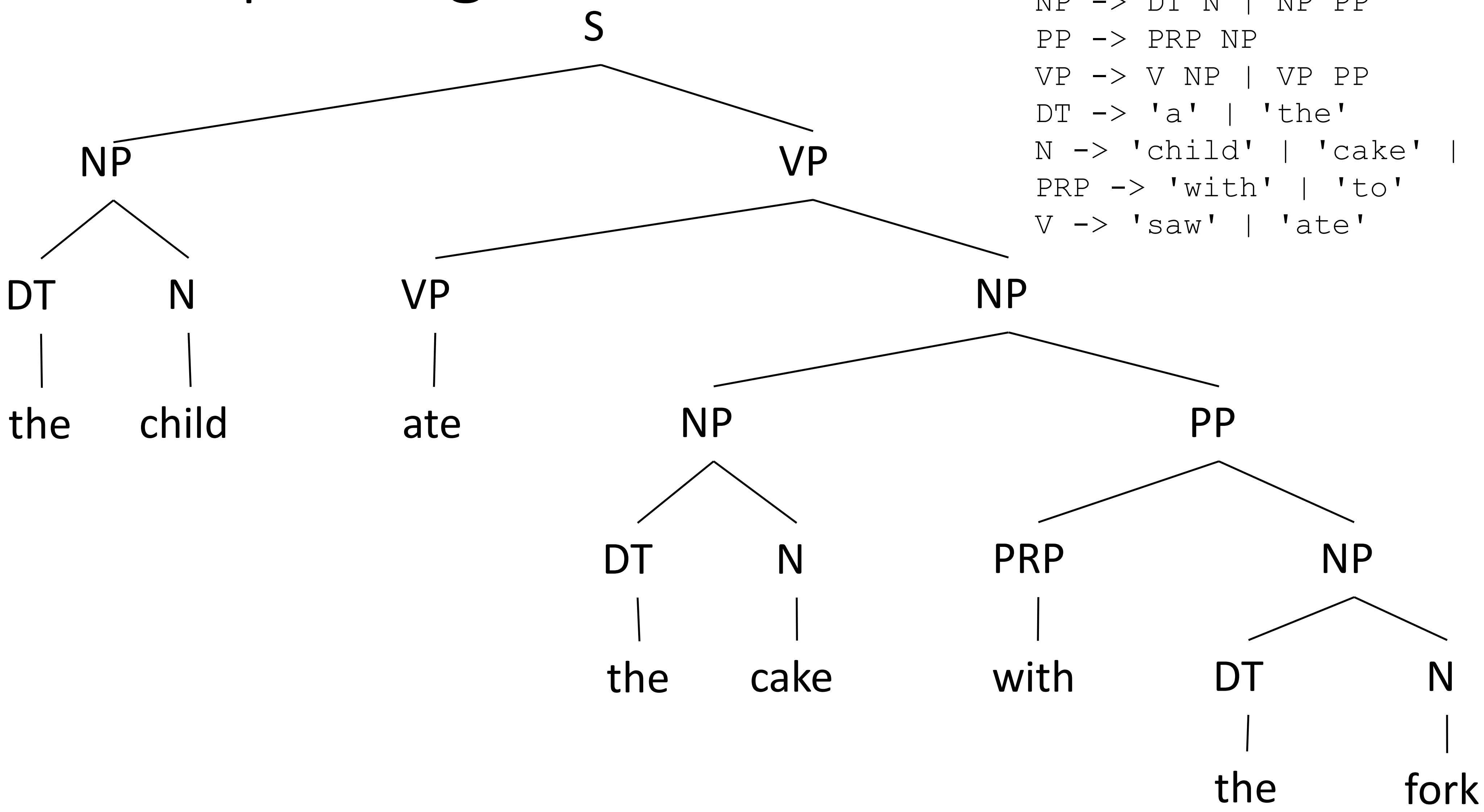
Top down parsing




```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

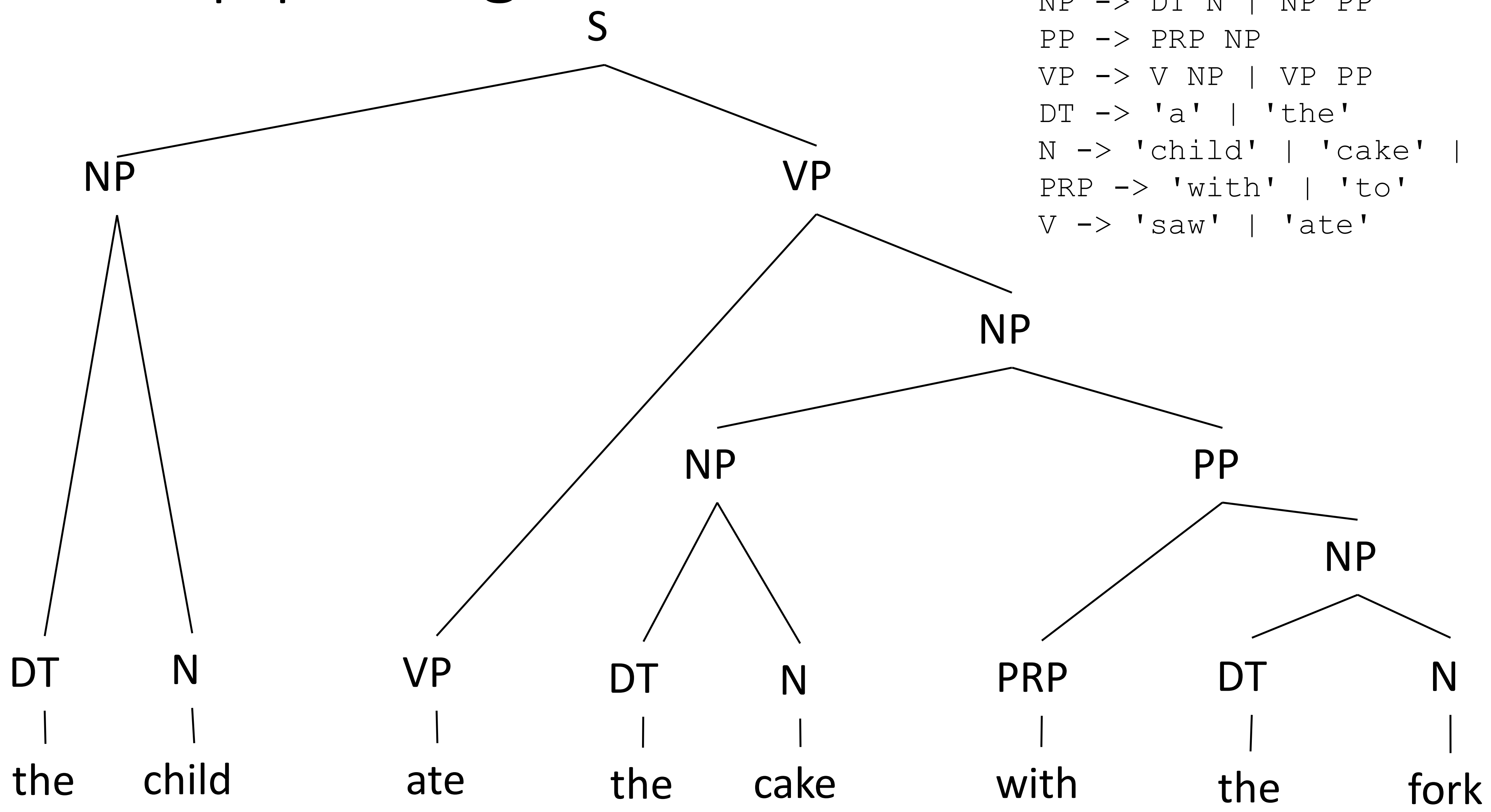
Top down parsing

```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```



Bottom up parsing

```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```



Bottom up vs. top down methods

- Bottom up
  - explores options that won’t lead to a full parse
  - Example: shift-reduce (srparser in nltk)
  - Example: CKY (Cocke-Kasami-Younger)
- Top down
  - explores options that don’t match the full sentence
  - Example: recursive descent (rdparser in nltk)
  - Example: Earley parser
- Dynamic programming
  - caches of intermediate results (memoization)

Introduction to NLP

Shift-Reduce Parsing

Shift-Reduce Parsing

- A bottom-up parser
  - Tries to match the RHS of a production until it can build an S
- *Shift* operation
  - Each word in the input sentence is pushed onto a stack
- *Reduce-n* operation
  - If the top *n* words on the top of the stack match the RHS of a production, then they are popped and replaced by the LHS of the production
- Breadth-first search
- Stopping condition
  - The process stops when the input sentence has been processed and S has been popped from the stack

Shift-Reduce Parsing Example

```
  [ * the child ate the cake]
S [ 'the' * child ate the cake]
R [ DT * child ate the cake]
S [ DT 'child' * ate the cake]
R [ DT N * ate the cake]
R [ NP * ate the cake]
S [ NP 'ate' * the cake]
R [ NP V * the cake]
S [ NP V 'the' * cake]
R [ NP V DT * cake]
S [ NP V DT 'cake' * ]
R [ NP V DT N * ]
R [ NP V NP * ]
R [ NP VP * ]
R [ S * ]
```

(S (NP (DT the) (N child)) (VP (V ate) (NP (DT the) (N cake))))

Shift-Reduce Parsing

- In nltk

```
>>> from nltk.app import srparser;
>>> srparser())
```

Introduction to NLP

244.

Cocke-Kasami-Younger (CKY) Parsing

Notes on Left Recursion

- Problematic for many parsing methods
  - Infinite loops when expanding
- But appropriate linguistically
  - NP -> DT N
  - NP -> PN
  - DT -> NP ‘s
  - Mary’s mother’s sister’s friend

Chart Parsing

- Top-down parsers have problems with expanding the same non-terminal
  - In particular, pre-terminals such as POS
  - Bad idea to use top-down (recursive descent) parsing as is
- Bottom-up parsers have problems with generating locally feasible subtrees that are not viable globally
- Chart parsing will address these issues

Dynamic Programming

- Motivation
  - A lot of the work is repeated
  - Caching intermediate results improves the complexity
- Dynamic programming
  - Building a parse for a substring [i,j] based on all parses [i,k] and [k, j] that are included in it.
- Complexity
  - O(n^3) for recognizing an input string of length *n*

Dynamic Programming

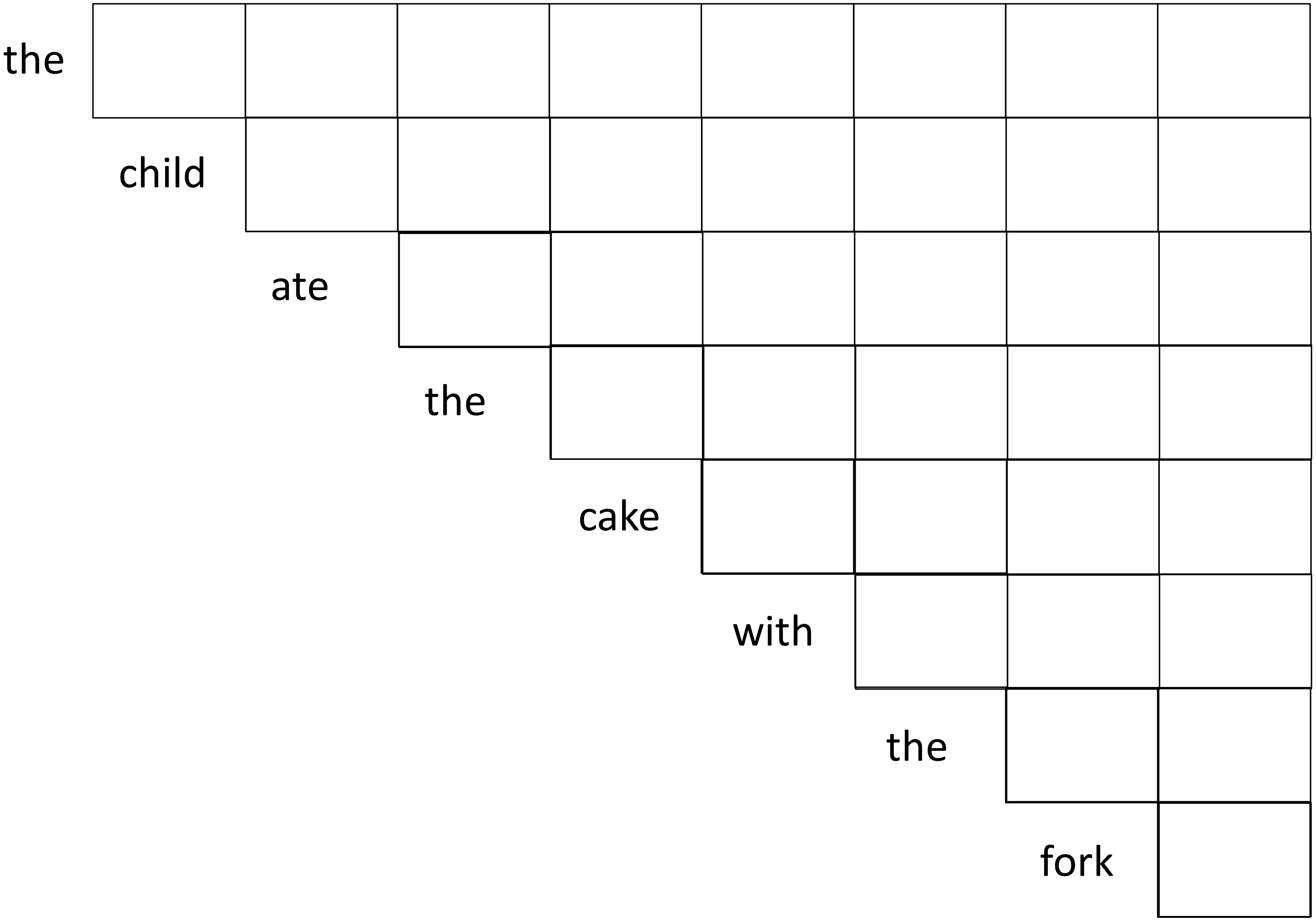
- CKY (Cocke-Kasami-Younger)
  - bottom-up
  - requires a normalized (binarized) grammar
- Earley parser
  - top-down
  - more complicated
  - (separate lecture)

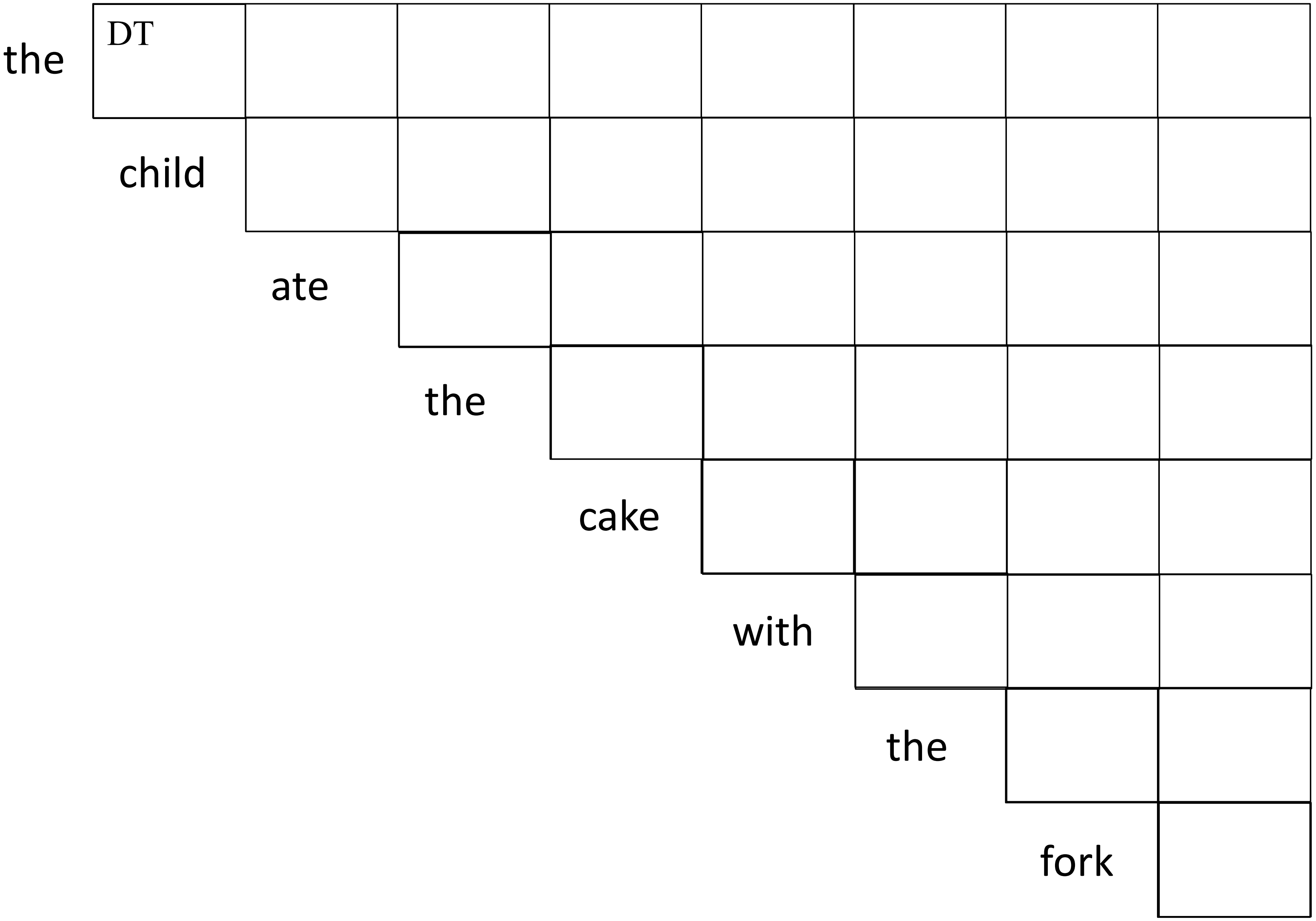
CKY Algorithm

```
function cky (sentence W, grammar G) returns table
  for i in 1..length(W) do
    table[i-1,i] = {A|A->Wi in G}
  for j in 2..length(W) do
    for i in j-2 down to 0 do
      for k in (i+1) to (j-1) do
        table[i,j] = table[i,j] union {A|A->BC in G, B in
table [I,k], C in table [k,j]}

If the start symbol S is in table [0,n] then W is in L(G)
```

Example

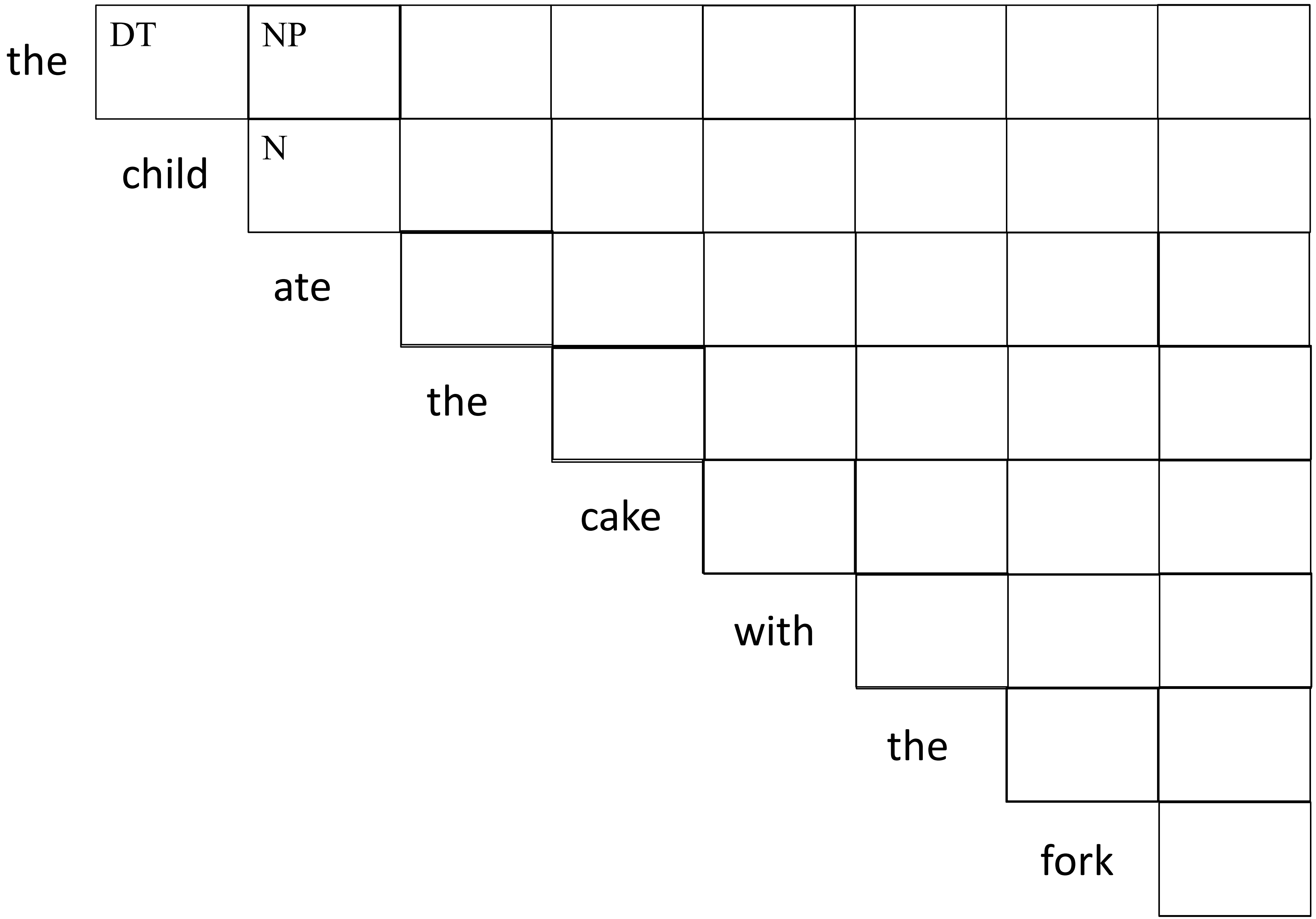
```
["the", "child", "ate", "the", "cake", "with", "the", "fork"]

    S -> NP VP
    NP -> DT N | NP PP
    PP -> PRP NP
    VP -> V NP | VP PP
    DT -> 'a' | 'the'
    N -> 'child' | 'cake' | 'fork'
    PRP -> 'with' | 'to'
    V -> 'saw' | 'ate'
```

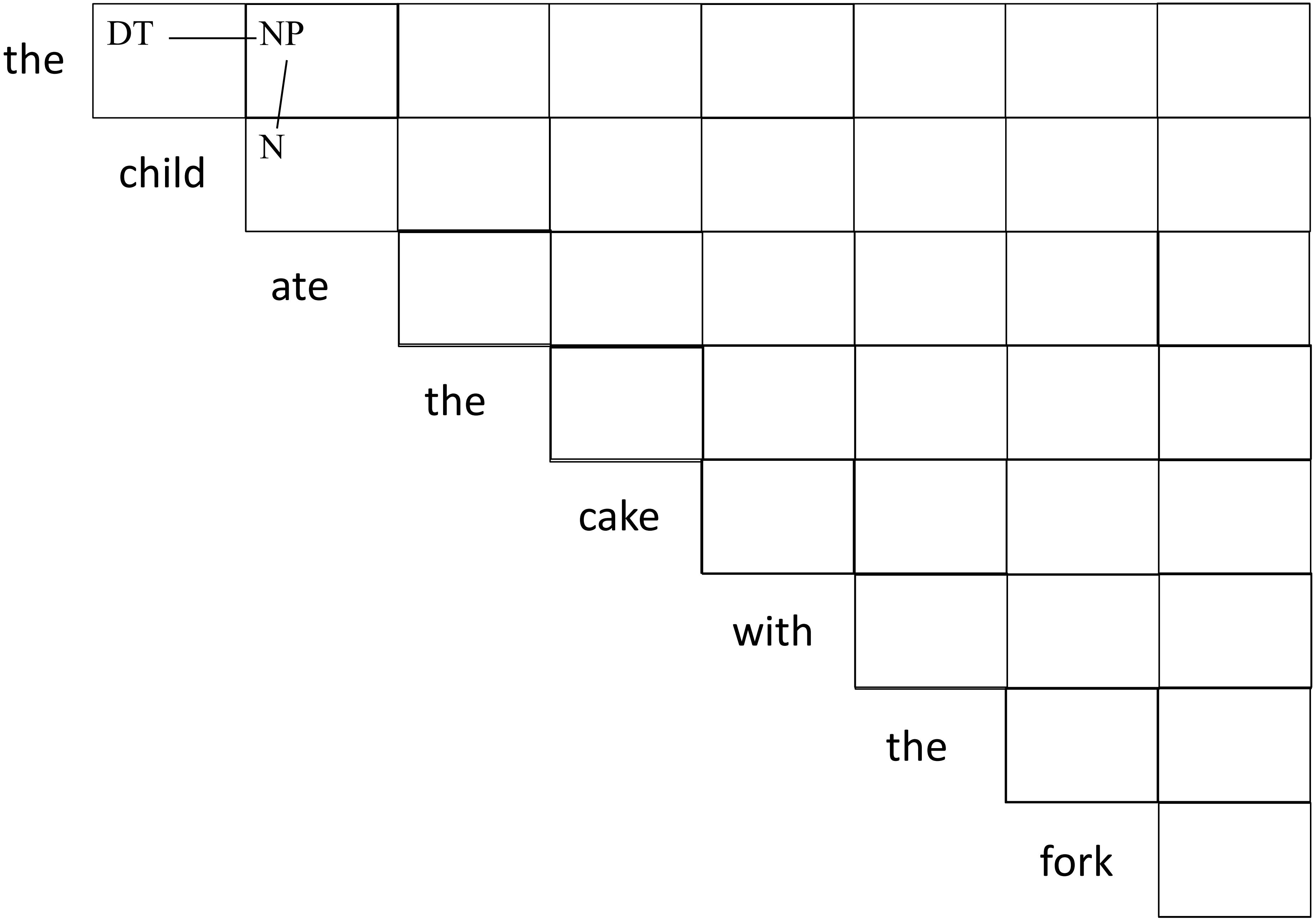

the
child
ate
the
cake
with
the
fork



the
DT
child
ate
the
cake
with
the
fork

the	DT							
	child	N						
		ate						
			the					
				cake				
					with			
						the		
							fork	


the
DT
NP
child
N
ate
the
cake
with
the
fork

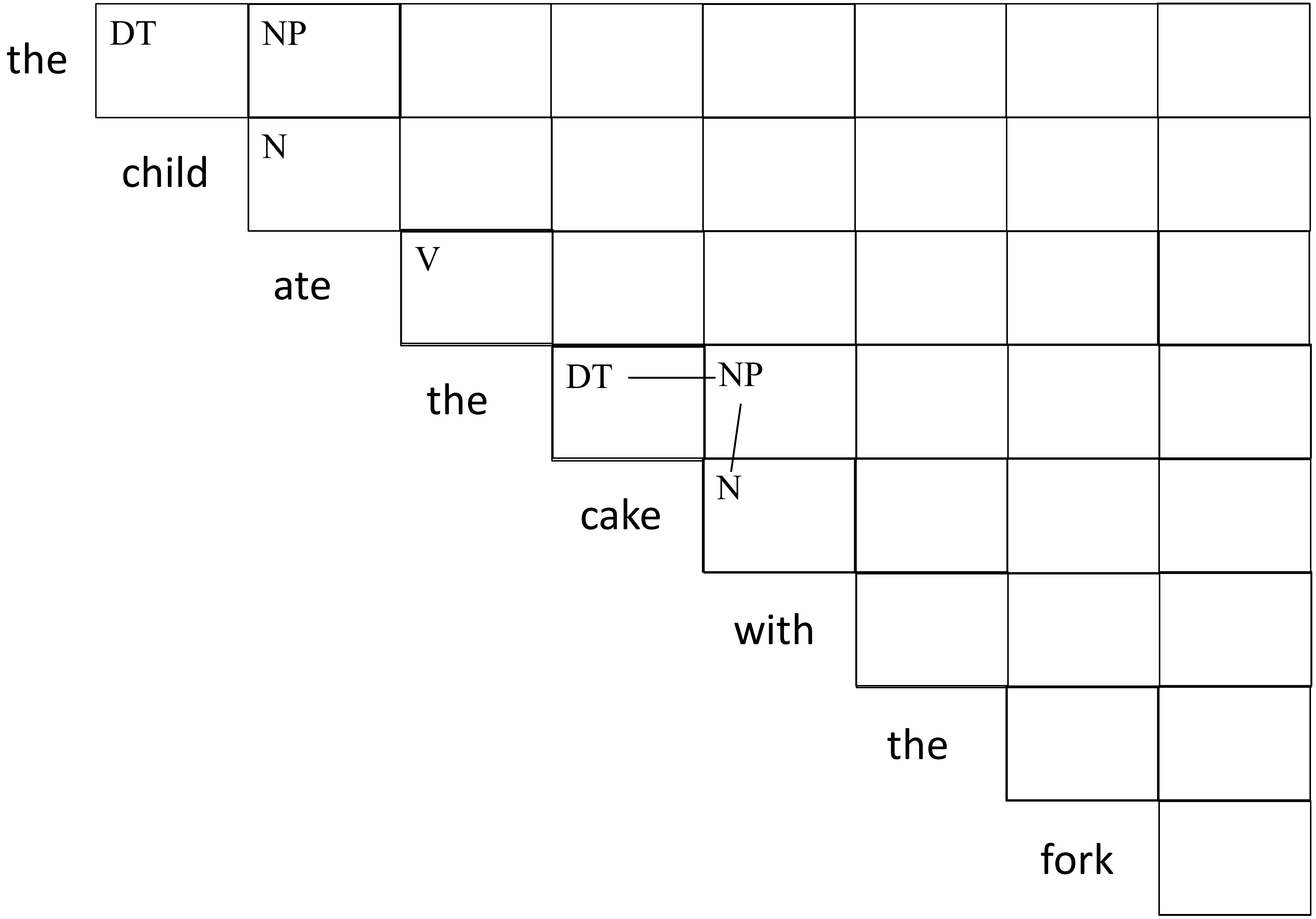

the
DT
NP
N
child
ate
the
cake
with
the
fork

the	DT	NP						
child		N						
ate			V					
the								
cake								
with								
the								
fork								

the	DT	NP						
child		N						
ate			V					
the				DT				
cake								
with								
the								
fork								

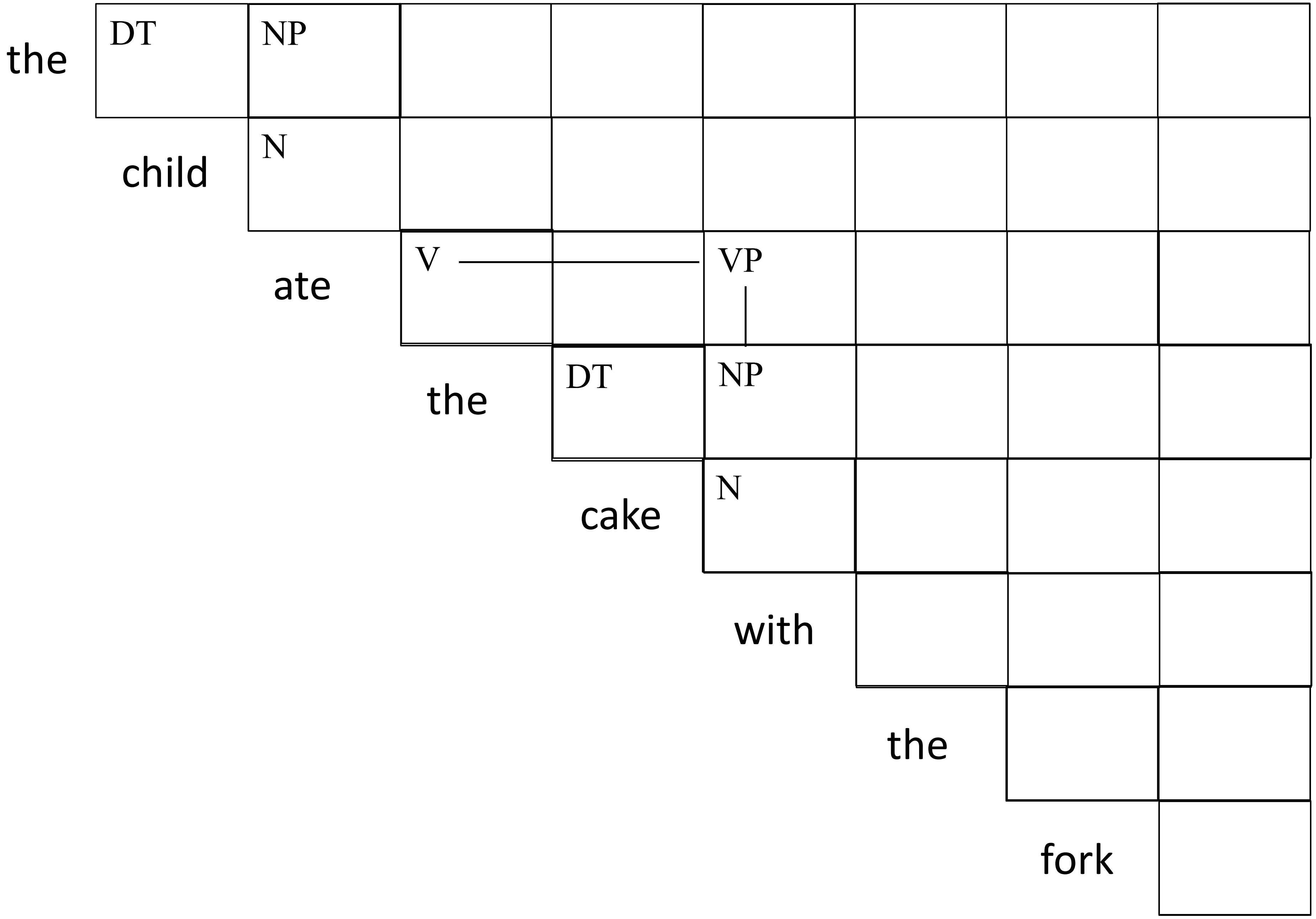
the	DT	NP						
child		N						
ate			V					
the				DT				
cake					N			
with								
the								
fork								

the	DT	NP						
child		N						
ate			V					
the				DT	NP			
cake					N			
with								
the								
fork								



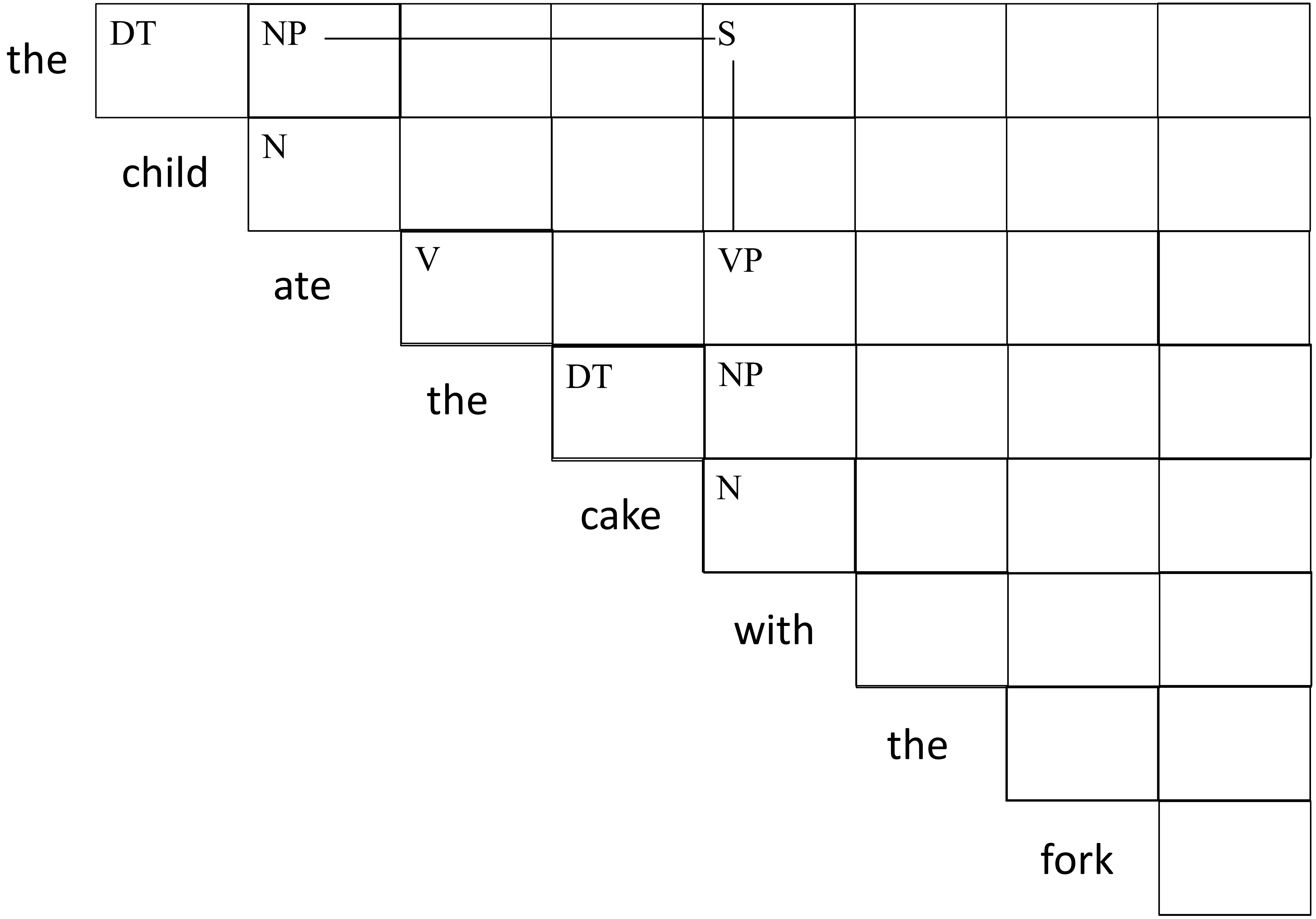
the
DT
NP
child
N
ate
V
the
DT
NP
cake
N
with
the
fork

the	DT	NP						
	child	N						
		ate	V		VP			
			the	DT	NP			
				cake	N			
					with			
						the		
							fork	



the
DT
NP
child
N
ate
V
VP
the
DT
NP
cake
N
with
the
fork

the	DT	NP			S			
child		N						
ate			V		VP			
the				DT	NP			
cake					N			
with								
the								
fork								


the
DT
NP
S
child
N
ate
V
VP
the
DT
NP
cake
N
with
the
fork

the	DT	NP			S			
	child	N						
		ate	V		VP			
			the	DT	NP			
				cake	N			
					with	PRP		
						the		
							fork	

the	DT	NP			S			
	child	N						
		ate	V		VP			
			the	DT	NP			NP
				cake	N			
					with	PRP		PP
						the	DT	NP
							fork	N

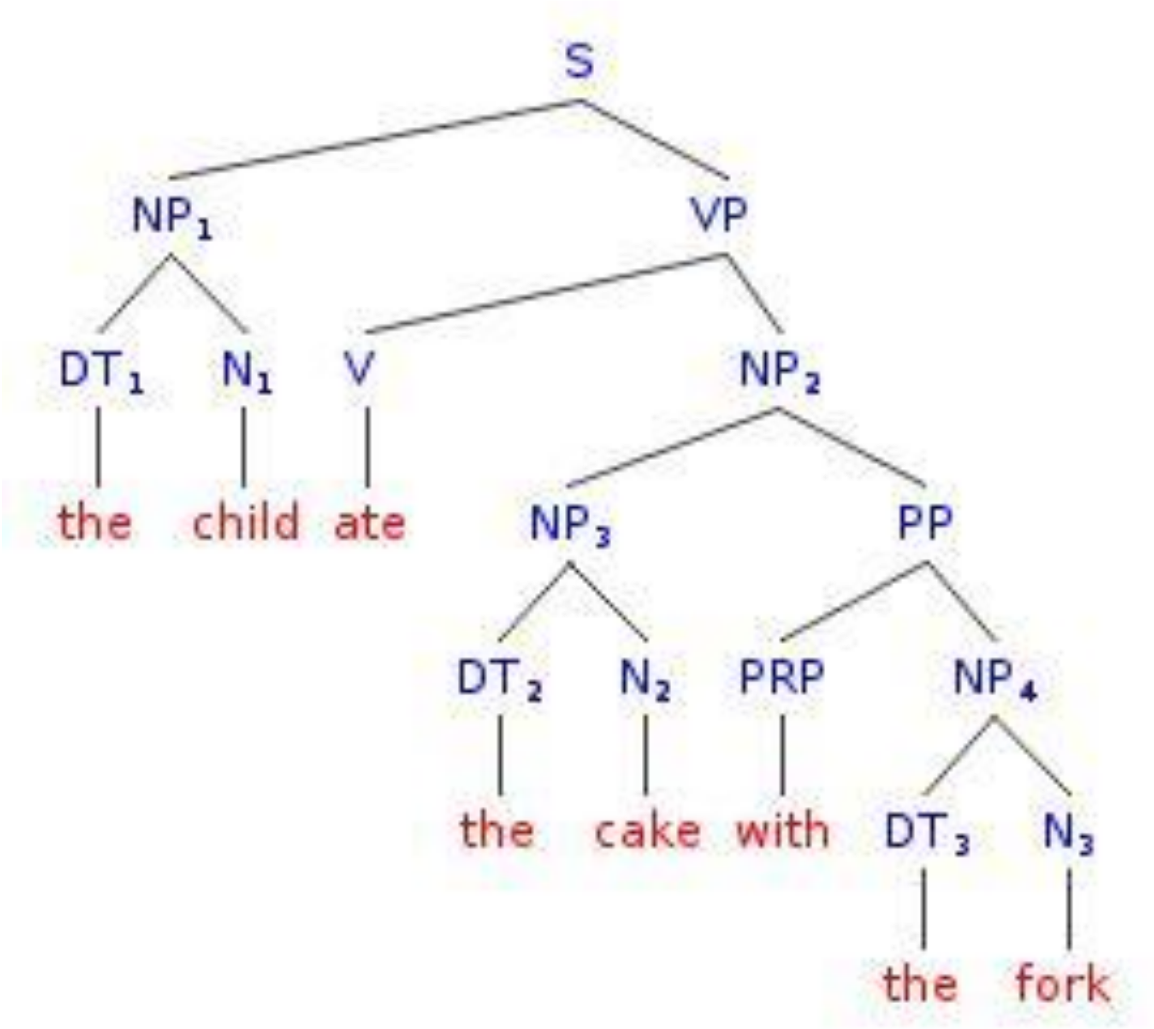
the	DT	NP			S			
	child	N						
		ate	V		VP			VP
			the	DT	NP			NP
				cake	N			
					with	PRP		PP
						the	DT	NP
							fork	N

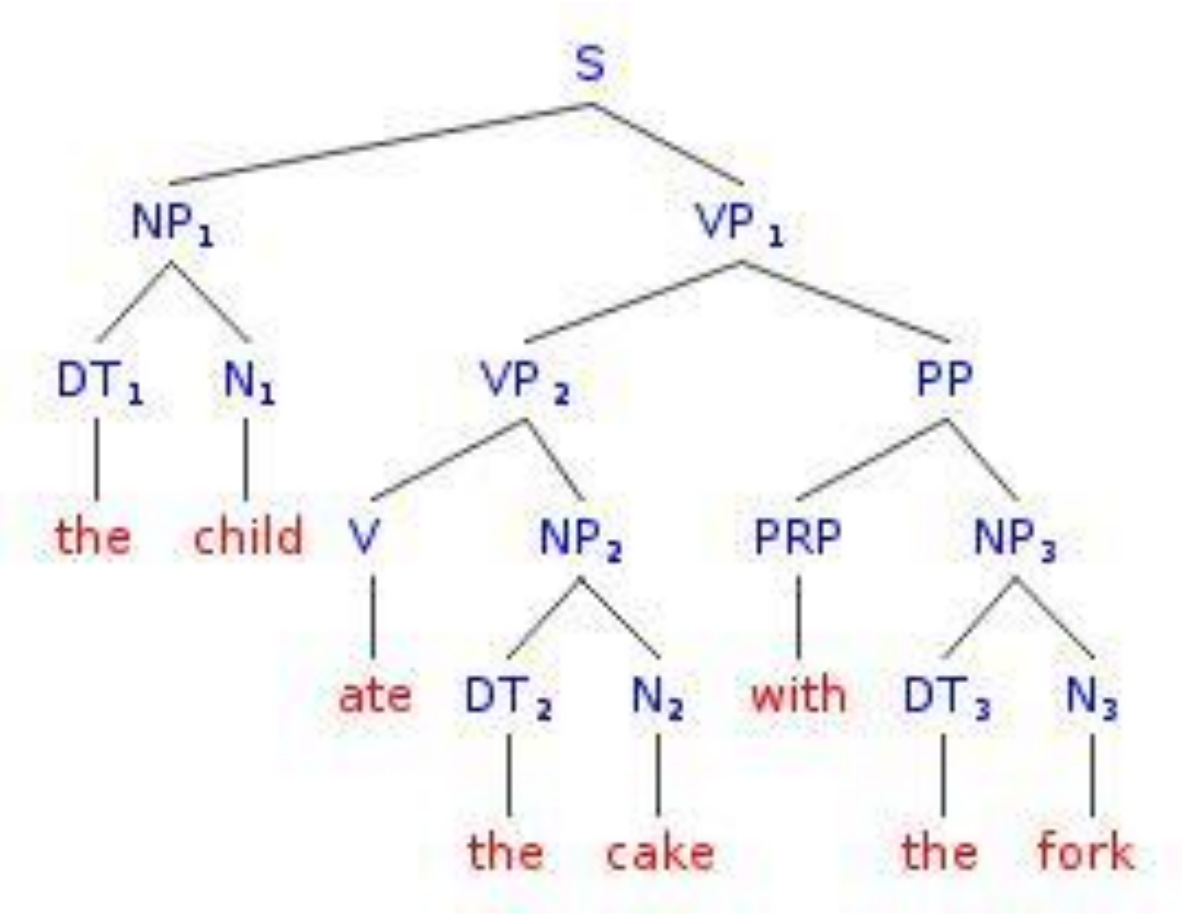
the	DT	NP			S			
	child	N						
		ate	V		VP			VP
			the	DT	NP			NP
				cake	N			
					with	PRP		PP
						the	DT	NP
							fork	N

the	DT	NP			S			S
child		N						
ate			V		VP			VP
the				DT	NP			NP
cake					N			
with						PRP		PP
the							DT	NP
fork								N

	1	2	3	4	5	6	7	8
the	DT	NP			S			S
child		N						
ate			V		VP			VP
the				DT	NP			NP
cake					N			
with						PRP		PP
the							DT	NP
fork								N

[0] DT [1] N [2] ==> [0] NP [2]
[3] DT [4] N [5] ==> [3] NP [5]
[6] DT [7] N [8] ==> [6] NP [8]
[2] V [3] NP [5] ==> [2] VP [5]
[5] PRP [6] NP [8] ==> [5] PP [8]
[0] NP [2] VP [5] ==> [0] S [5]
[3] NP [5] PP [8] ==> [3] NP [8]
[2] V [3] NP [8] ==> [2] VP [8]
[2] VP [5] PP [8] ==> [2] VP [8]
[0] NP [2] VP [8] ==> [0] S [8]





What is the *meaning* of each of these sentences?

```
(S
  (NP (DT the) (N child))
  (VP
    (VP (V ate) (NP (DT the) (N cake)))
    (PP (PRP with) (NP (DT the) (N fork)))))
```

```
(S
  (NP (DT the) (N child))
  (VP
    (VP (V ate) (NP (DT the) (N cake)))
    (PP (PRP with) (NP (DT the) (N fork)))))
```

```
(S
  (NP (DT the) (N child))
  (VP
    (V ate)
    (NP
      (NP (DT the) (N cake))
      (PP (PRP with) (NP (DT the) (N fork))))))
```

Complexity of CKY

- Space complexity
  - There are $O(n^2)$ cells in the table
- Single parse
  - Each cell requires a linear lookup.
  - Total time complexity is $O(n^3)$
- All parses
  - Total time complexity is exponential

A longer example

```
["take", "this", "book"]

    S -> NP VP | Aux NP VP | VP
    NP -> PRON | Det Nom
    Nom -> N | Nom N | Nom PP
    PP -> PRP NP
    VP -> V | V NP | VP PP
    Det -> 'the' | 'a' | 'this'
    PRON -> 'he' | 'she'
    N -> 'book' | 'boys' | 'girl'
    PRP -> 'with' | 'in'
    V -> 'takes' | 'take'
```

Non-binary productions

```
["take", "this", "book"]

    S -> NP VP | Aux NP VP | VP
    NP -> PRON | Det Nom
    Nom -> N | Nom N | Nom PP
    PP -> PRP NP
    VP -> V | V NP | VP PP
    Det -> 'the' | 'a' | 'this'
    PRON -> 'he' | 'she'
    N -> 'book' | 'boys' | 'girl'
    PRP -> 'with' | 'in'
    V -> 'takes' | 'take'
```

Chomsky Normal Form (CNF)

- All rules have to be in binary form:
  - $X \rightarrow Y\ Z$ or $X \rightarrow w$
- This introduces new non-terminals for
  - hybrid rules
  - n-ary rules
  - unary rules
  - epsilon rules (e.g., $NP \rightarrow \varepsilon$)
- Any CFG can be converted to CNF
  - See Aho & Ullman p. 152

ATIS grammar

Original version

S → NP VP
S → Aux NP VP

S → VP

NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

From Jurafsky and Martin

ATIS grammar in CNF

Original version

S → NP VP
S → Aux NP VP

S → VP

NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

CNF version

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
PP → Prep NP

ATIS grammar in CNF

Original version

S → NP VP
S → Aux NP VP

S → VP

NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

CNF version

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
PP → Prep NP

Chomsky Normal Form

- All rules have to be in binary form:
  - X $\rightarrow$ Y Z or X $\rightarrow$ w
- New non-terminals for hybrid rules, n-ary and unary rules:
  - INF-VP $\rightarrow$ to VP *becomes*
    - INF-VP $\rightarrow$ TO VP
    - TO $\rightarrow$ to
  - S $\rightarrow$ Aux NP VP *becomes*
    - S $\rightarrow$ R1 VP
    - R1 $\rightarrow$ Aux NP
  - S $\rightarrow$ VP VP $\rightarrow$ Verb VP $\rightarrow$ Verb NP VP $\rightarrow$ Verb PP *becomes*
    - S $\rightarrow$ book
    - S $\rightarrow$ buy
    - S $\rightarrow$ R2 PP
    - S $\rightarrow$ Verb PP
  - etc.

Issues with CKY

- Weak equivalence only
  - Same language, different structure
  - If the grammar had to be converted to CNF, then the final parse tree doesn’t match the original grammar
  - However, it can be converted back using a specific procedure
- Syntactic ambiguity
  - (Deterministic) CKY has no way to perform syntactic disambiguation

Notes

- Demo:
  - http://lxmls.it.pt/2015/cky.html
- Recognizing vs. parsing
  - Recognizing just means determining if the string is part of the language defined by the CFG
  - Parsing is more complicated – it involves producing a parse tree