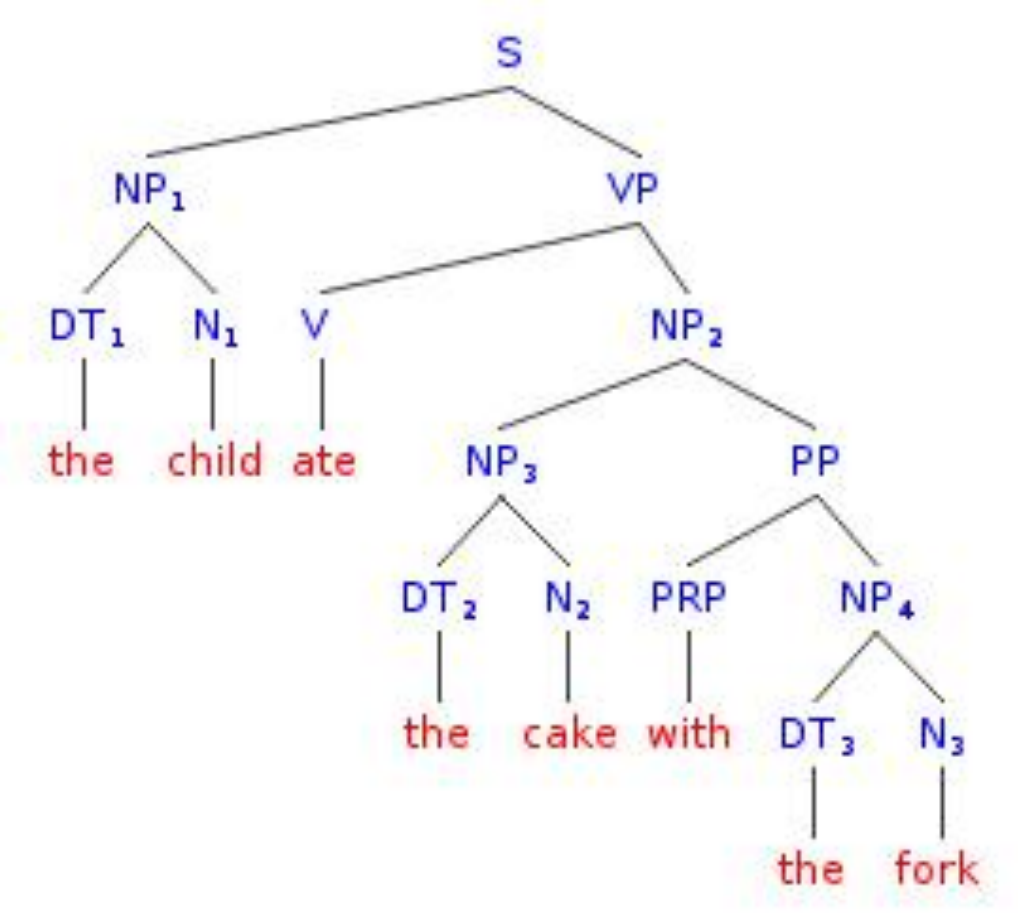
NLP

Introduction to NLP

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

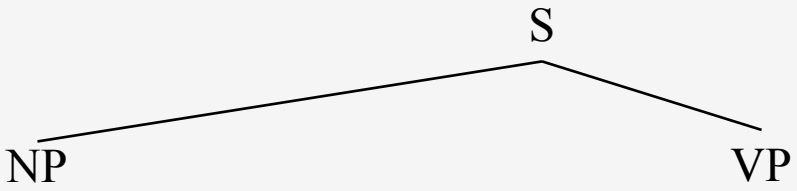
- There are two types of constraints on the parses
  - From the input sentence
  - From the grammar
- Therefore, two general approaches to parsing
  - Top-down
  - Bottom-up

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

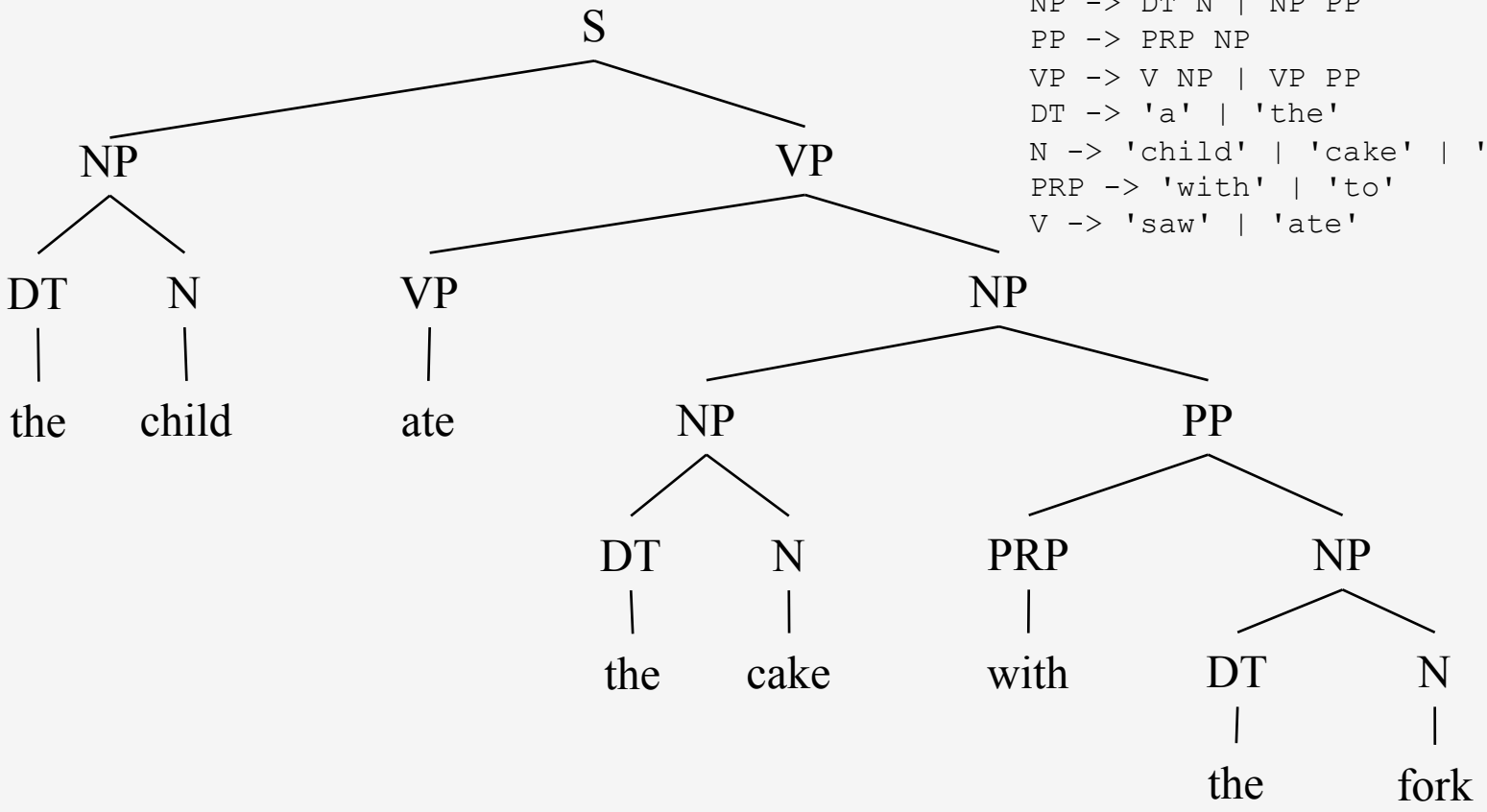

```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

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'
```

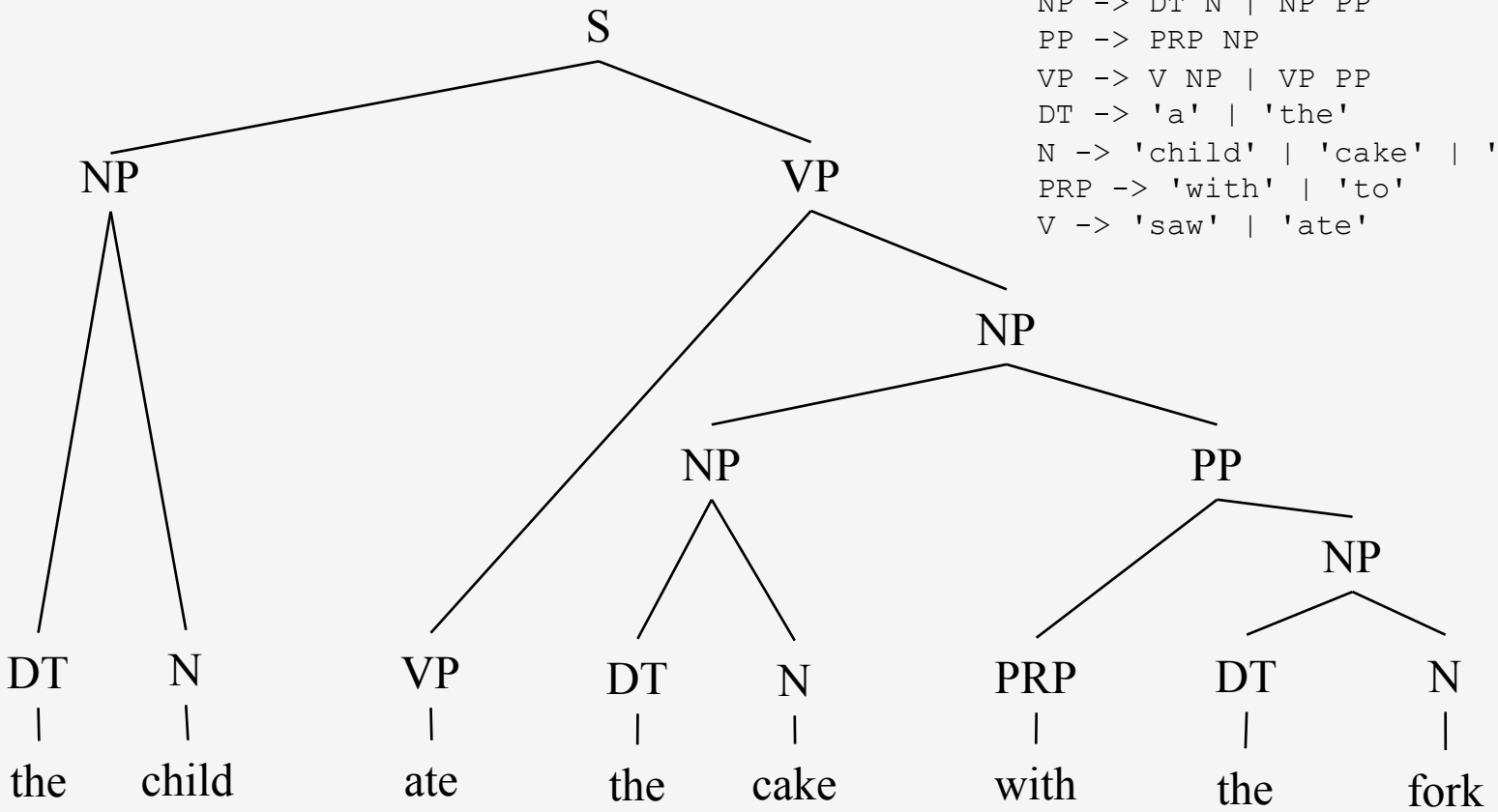
Top Down Parsing

```
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.
- Top down
  - explores options that don't match the full sentence.
- Dynamic programming
  - caches of intermediate results (memoization)
- Cocke-Kasami-Younger (CKY) parser
  - based on dynamic programming

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* 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
- 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]
```

Shift-reduce Parsing Example

```
  [ * the child ate the cake]
S [ 'the' * child ate the cake]
```

Shift-reduce Parsing Example

```
  [ * the child ate the cake]
S [ 'the' * child ate the cake]
R [ DT * child ate the cake]
```

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]
```

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]
```

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' * ]
```

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 * ]
```

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))))

NLP

Introduction to NLP

Cocke-Kasami-Younger (CKY) Parsing

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

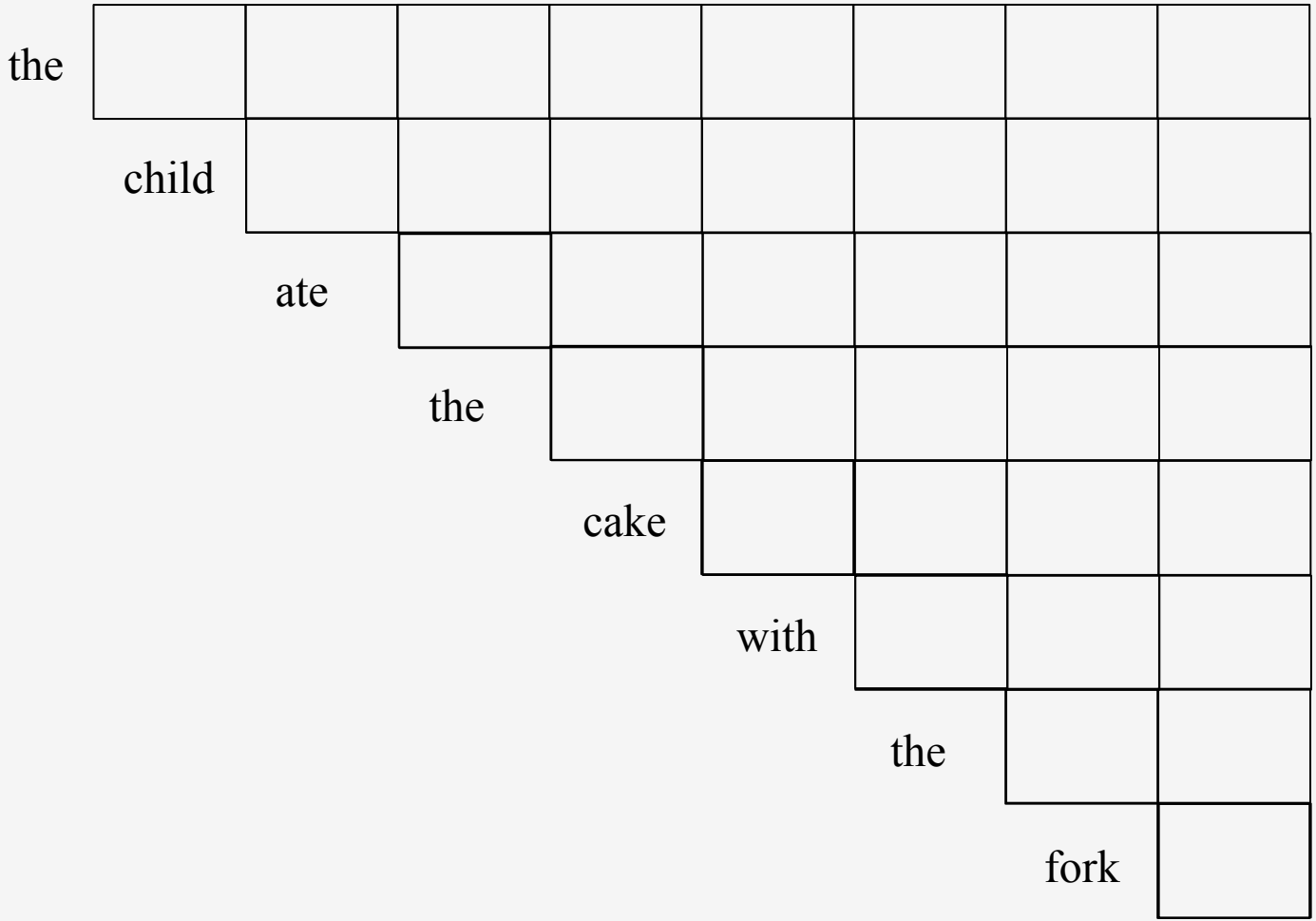
Dynamic Programming

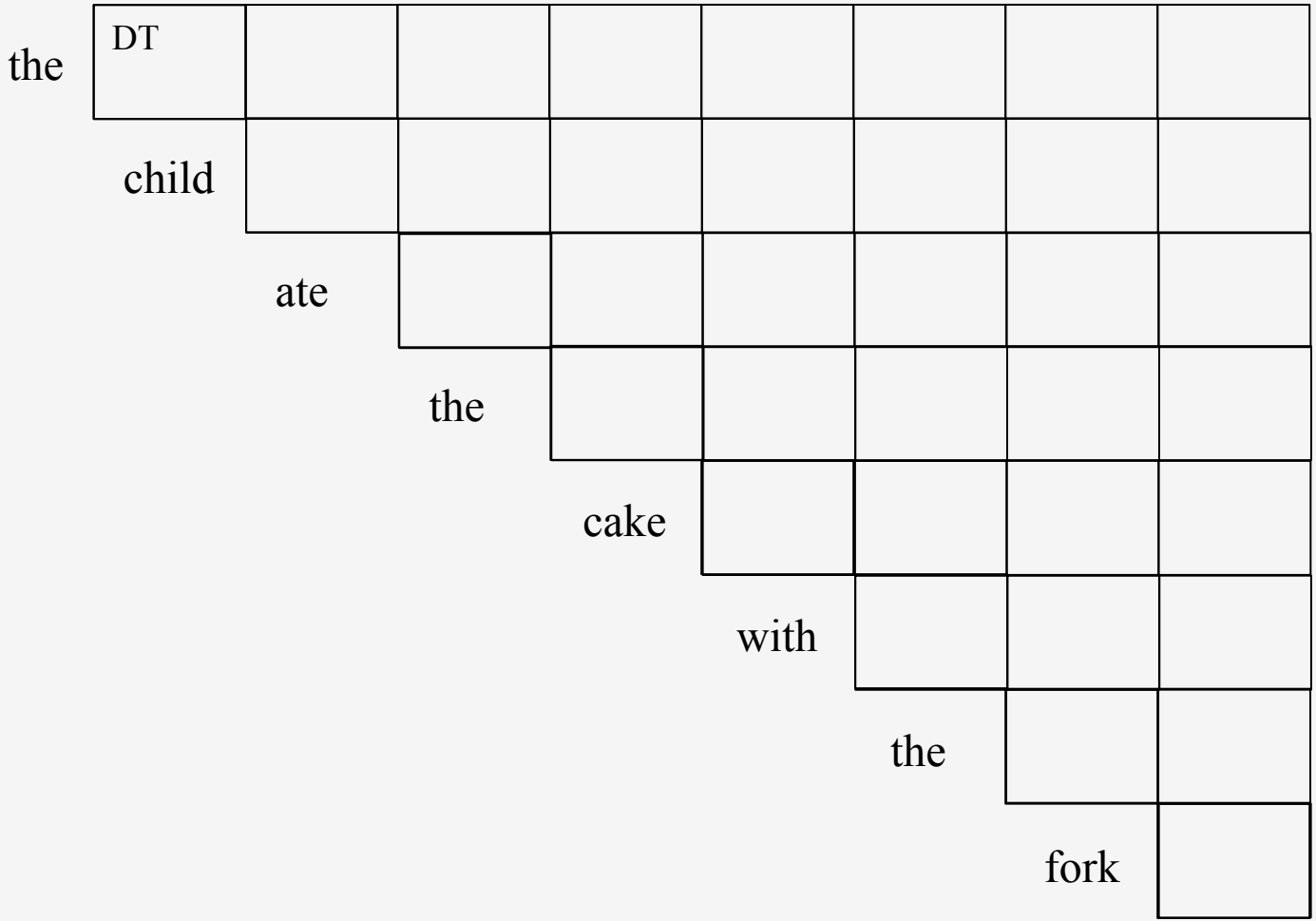
- CKY (Cocke-Kasami-Younger)
  - bottom-up
  - requires a normalized (binarized) grammar
- Earley parser
  - top-down
  - more complicated

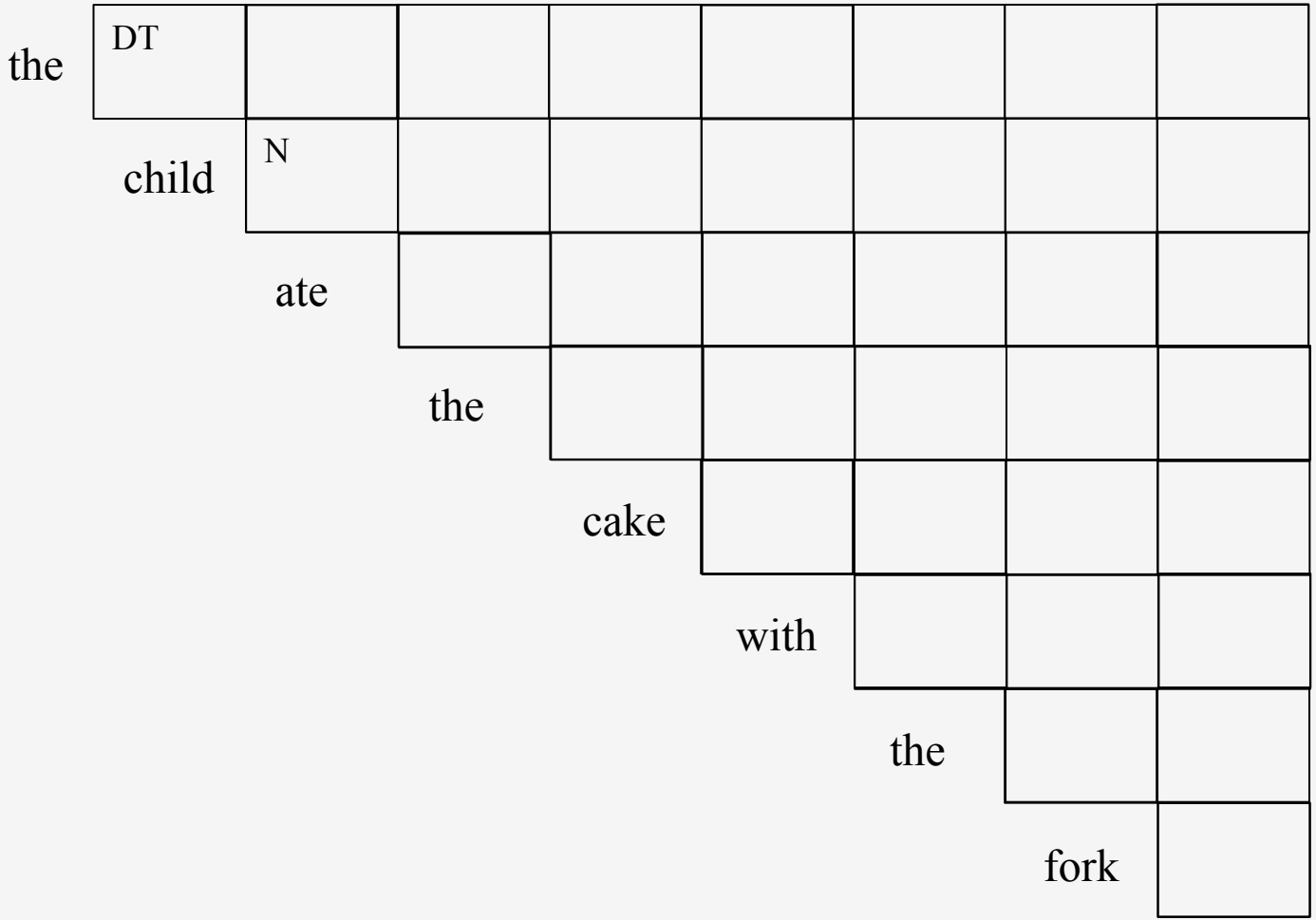
Example

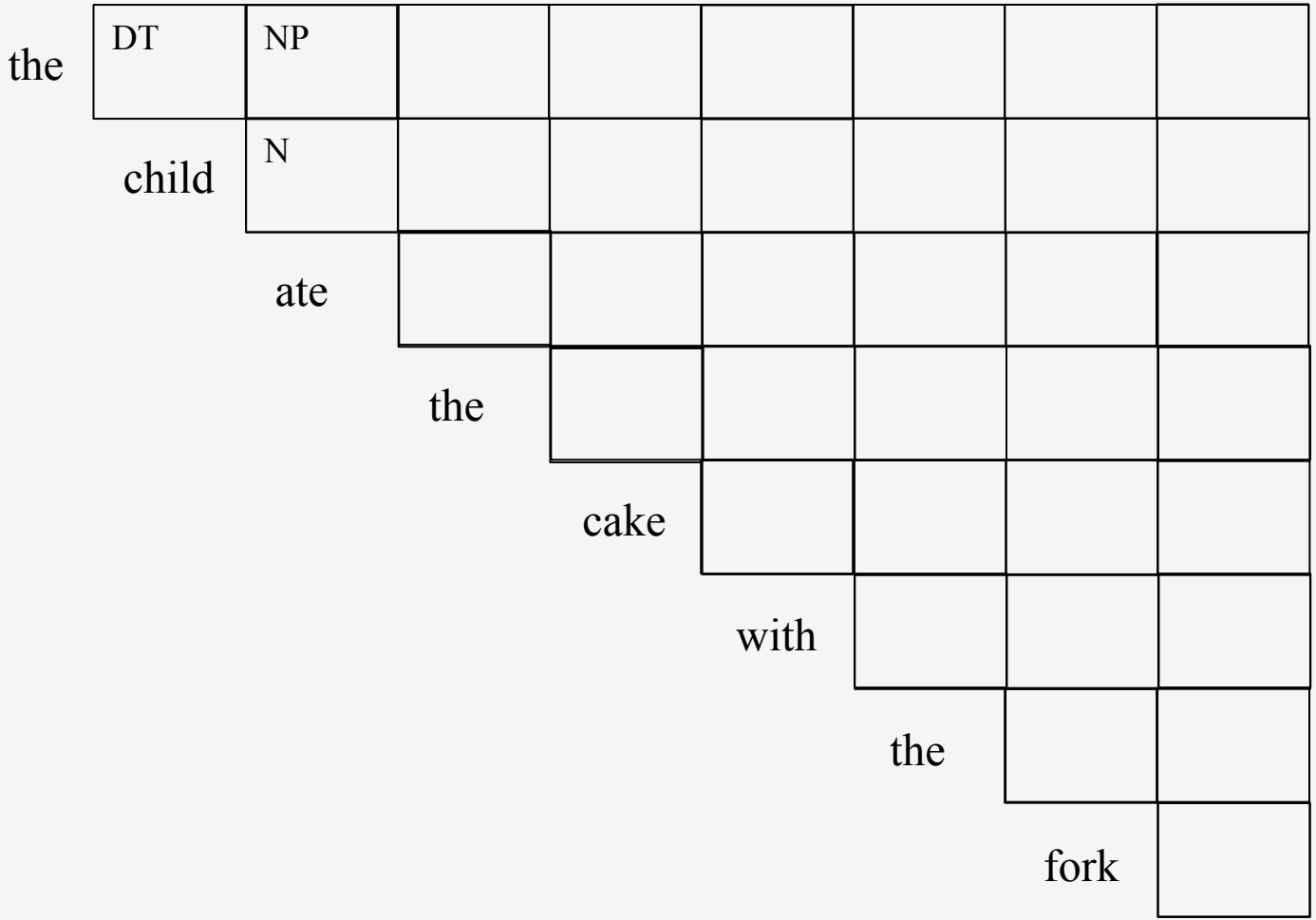
```
["the", "child", "ate", "the", "cake", "with", "the", "fork"]
```

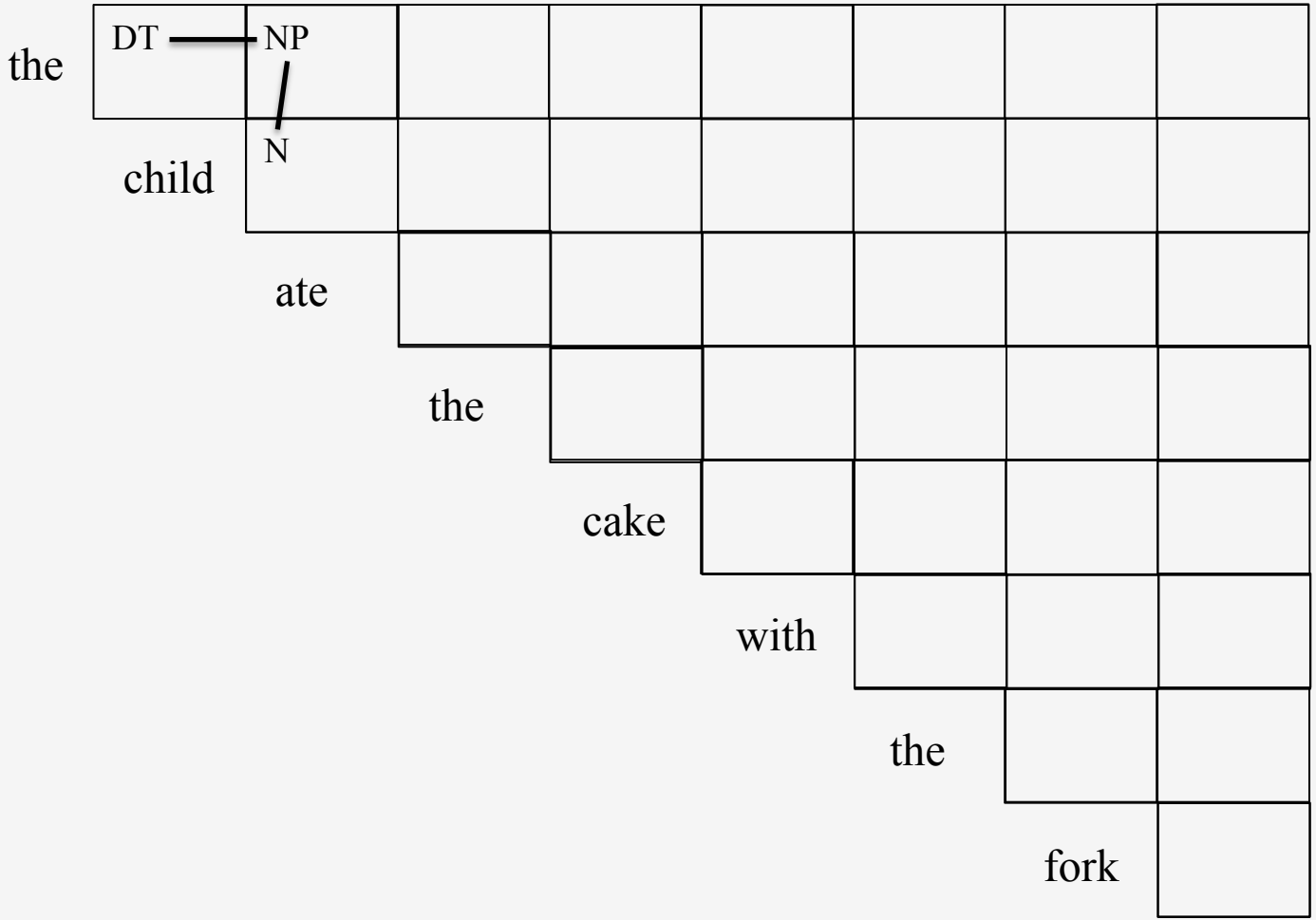
```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

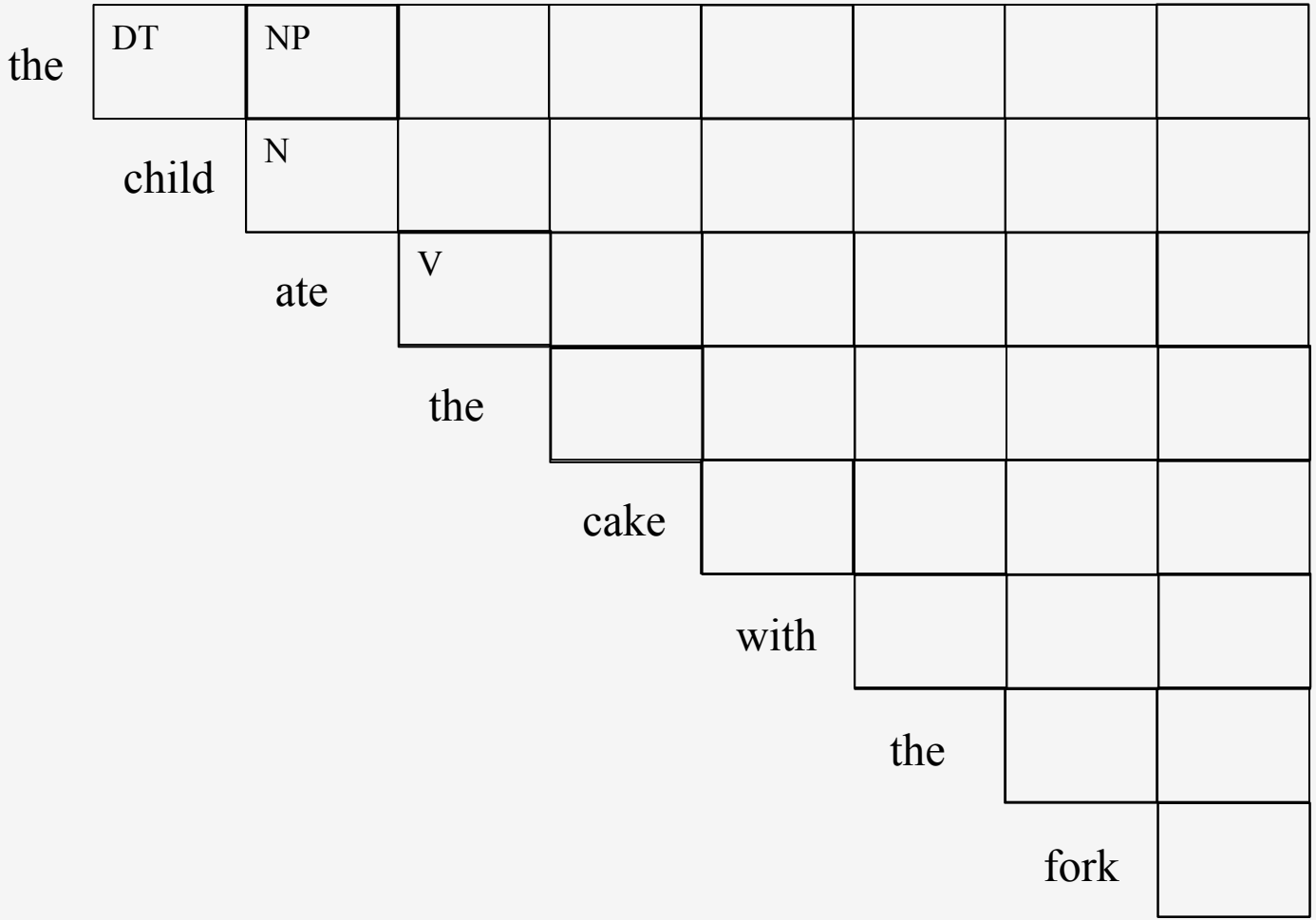

the
child
ate
the
cake
with
the
fork

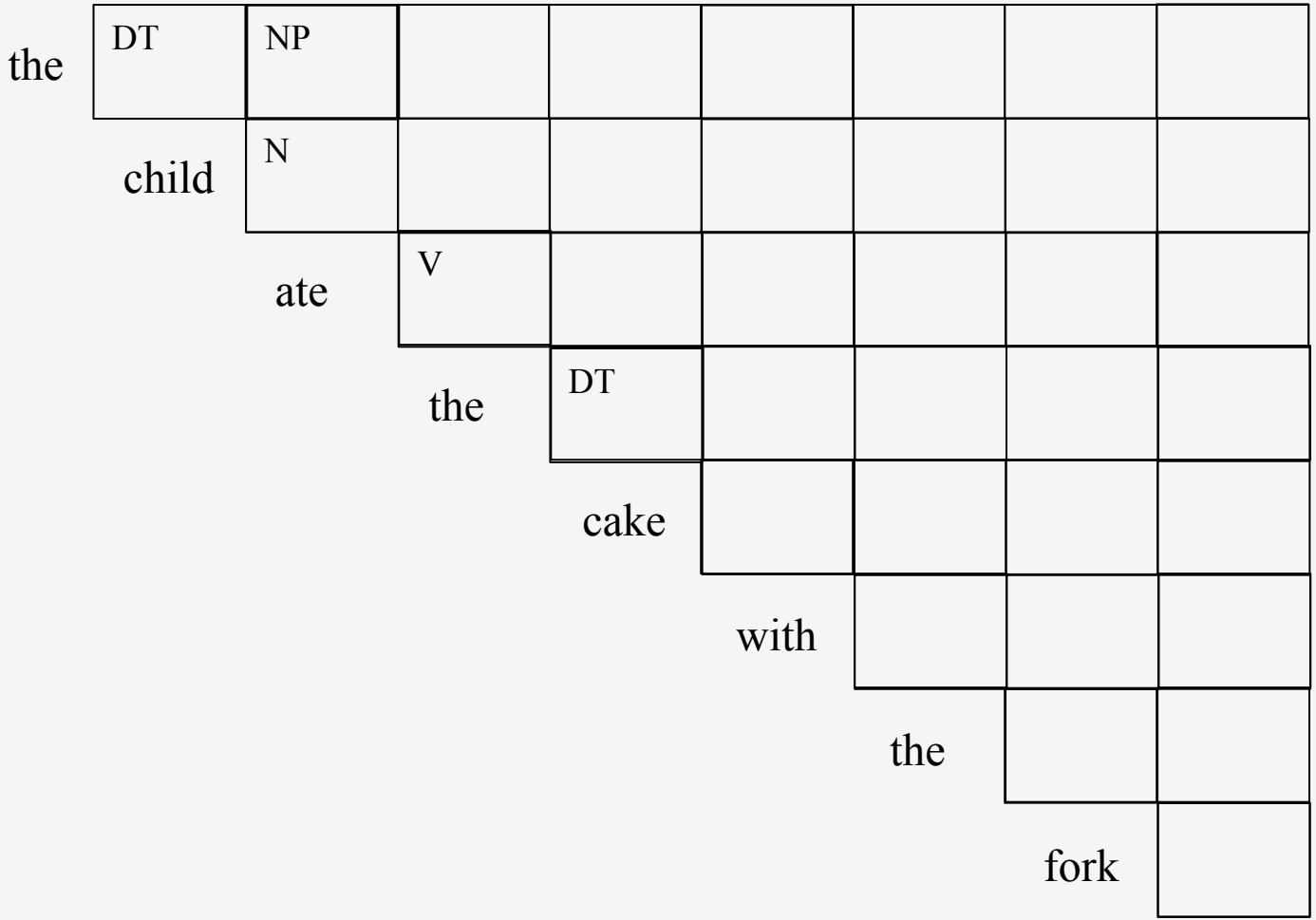

the
DT
child
ate
the
cake
with
the
fork

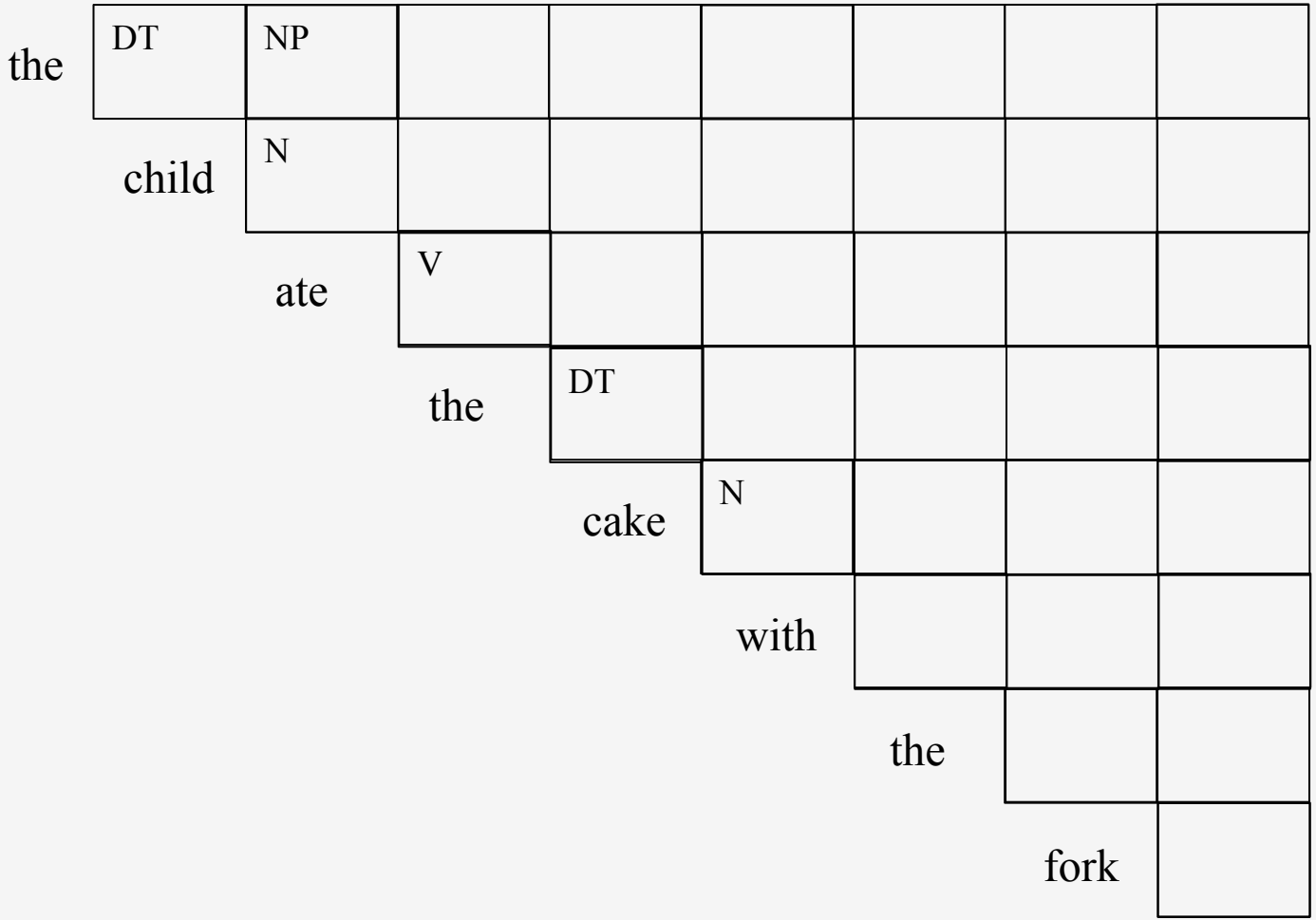

the
DT
child
N
ate
the
cake
with
the
fork

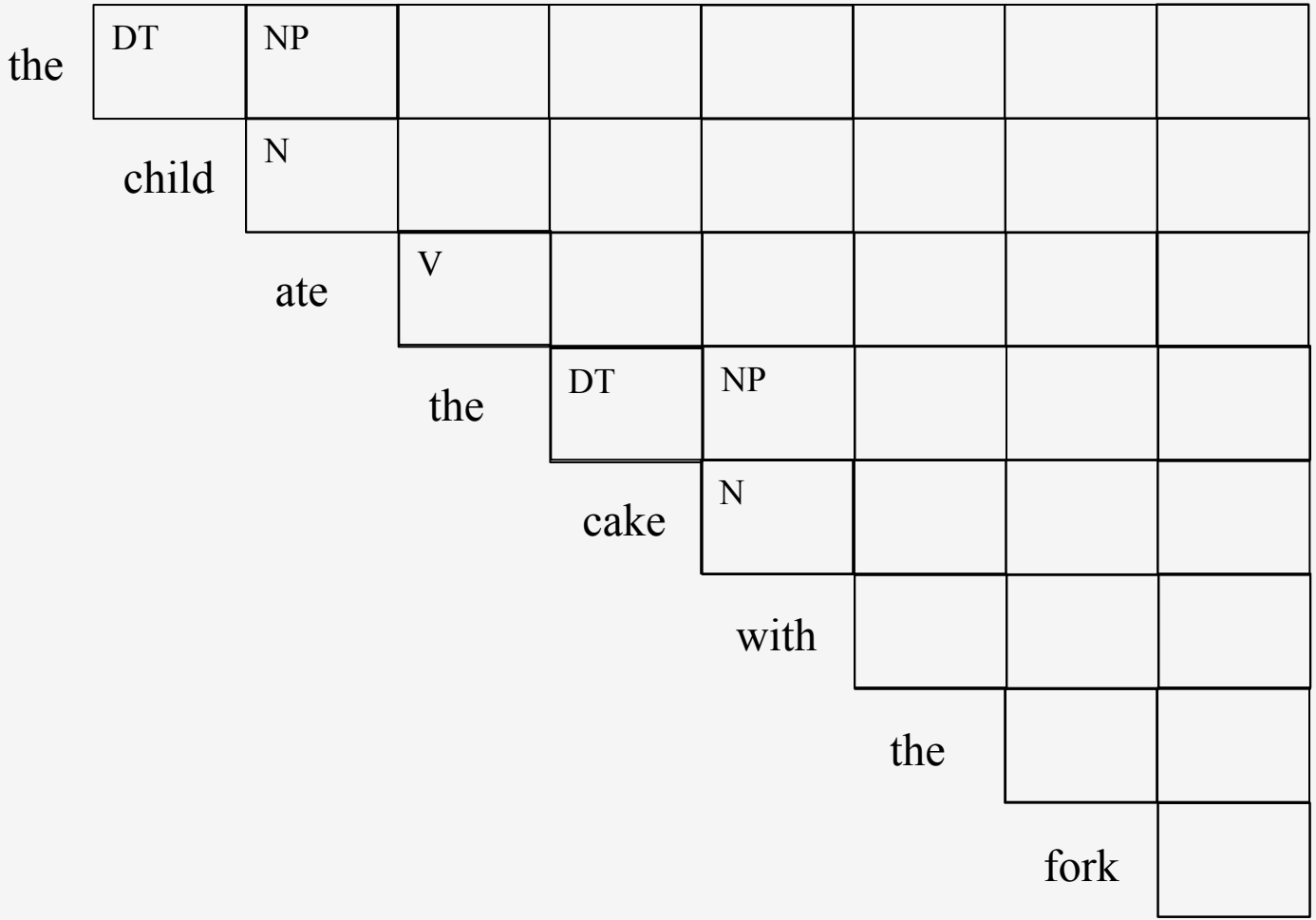

the
DT
NP
child
N
ate
the
cake
with
the
fork

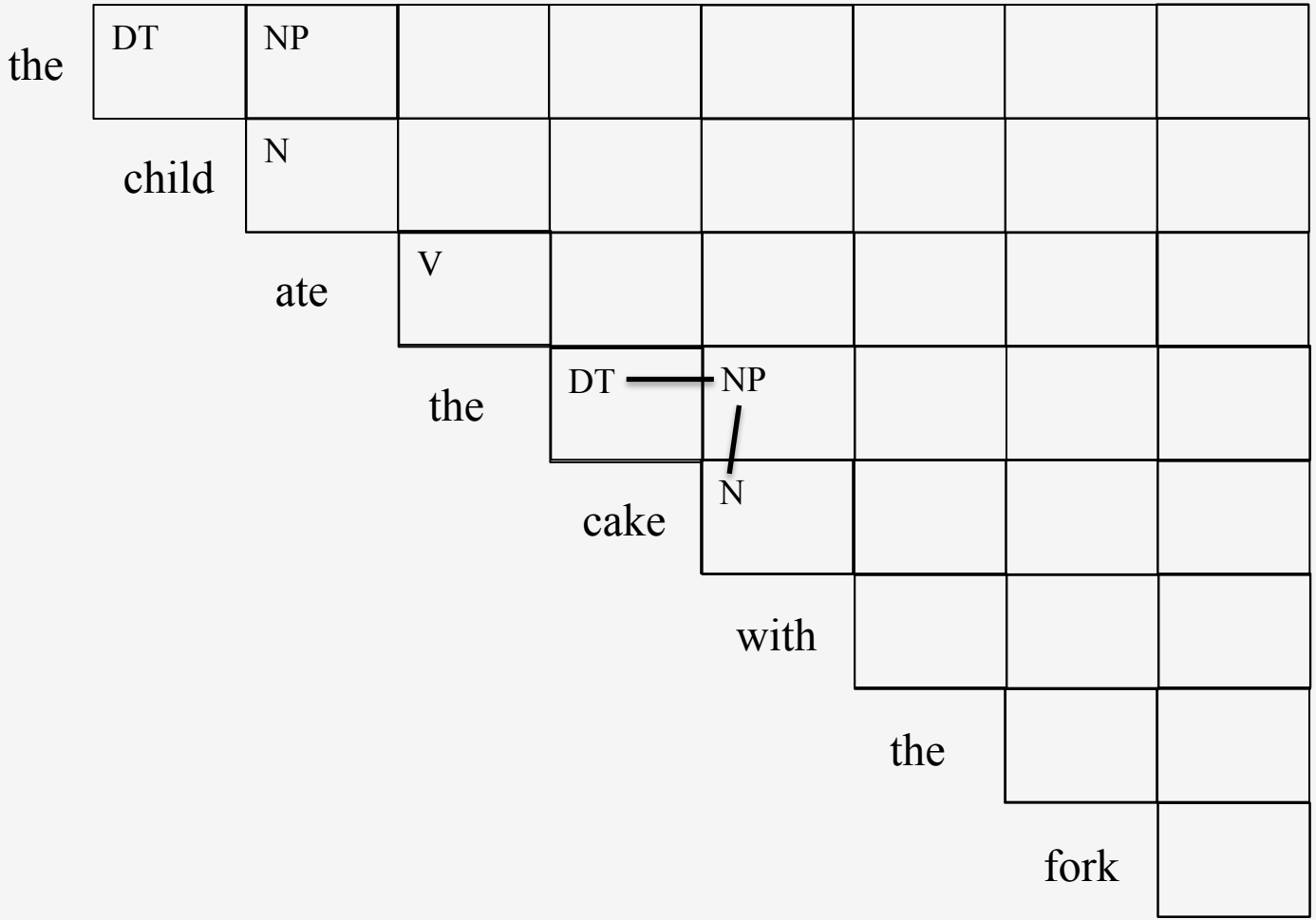

the
DT
NP
child
N
ate
the
cake
with
the
fork

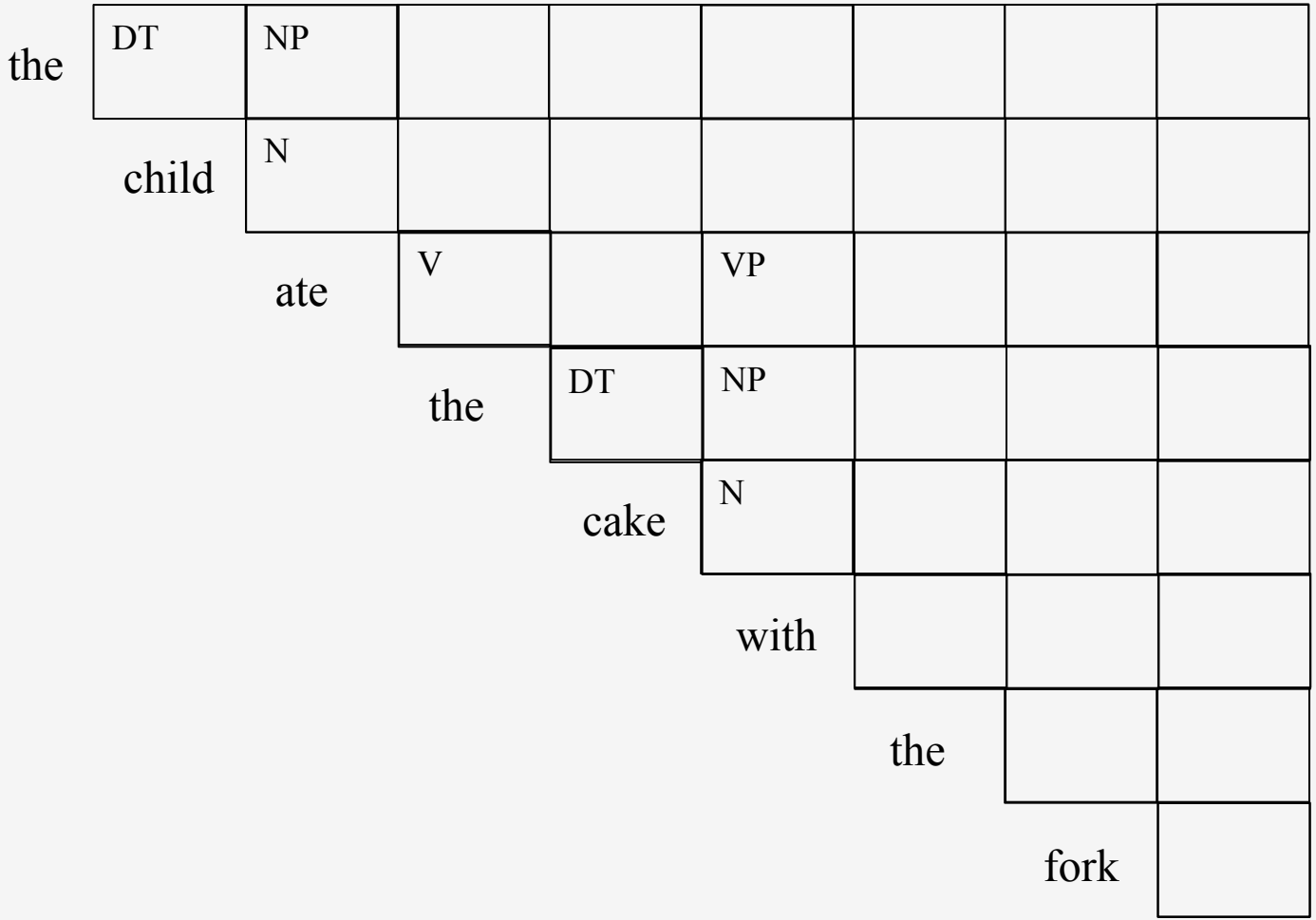

the
DT
NP
child
N
ate
V
the
cake
with
the
fork

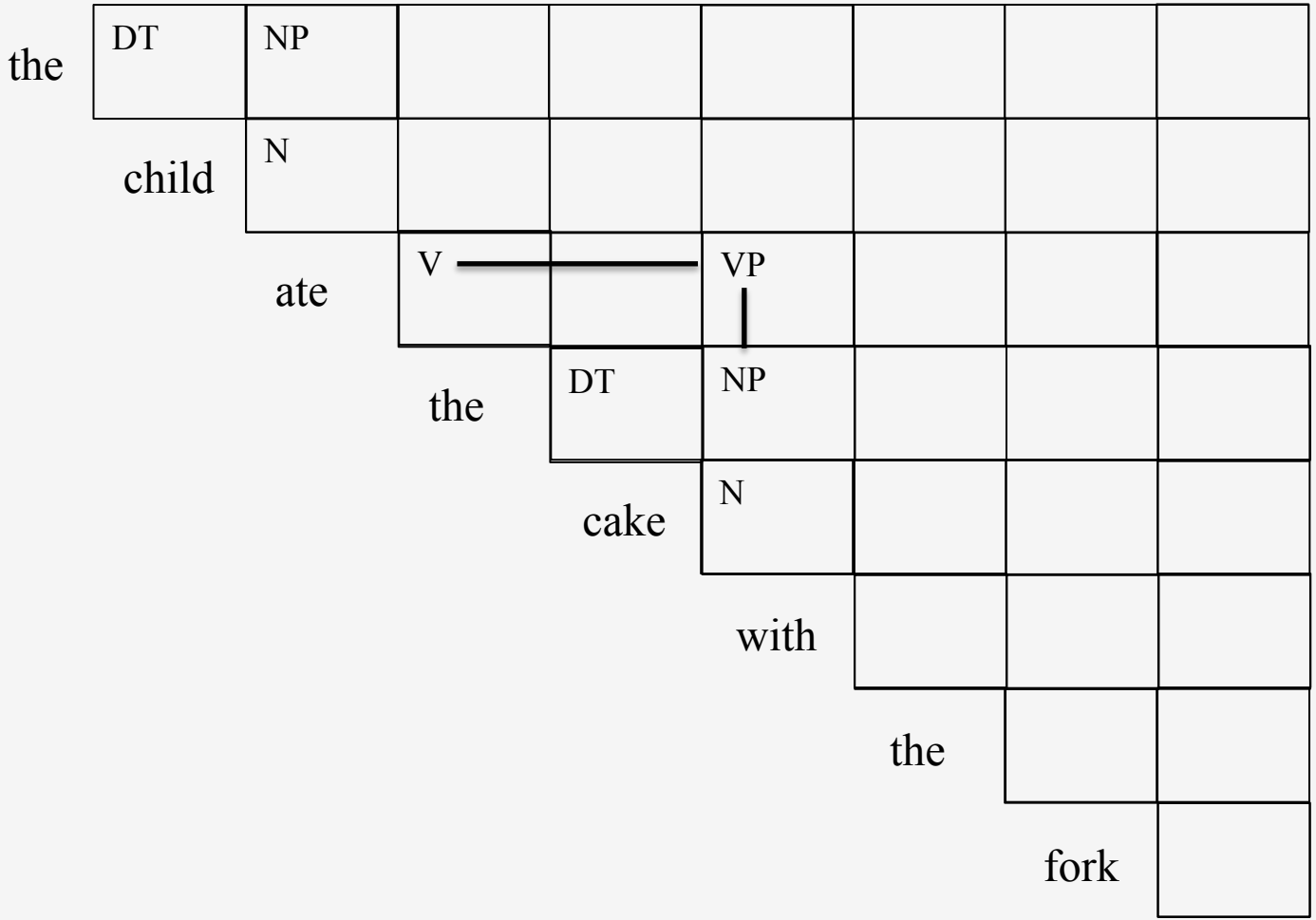

the
DT
NP
child
N
ate
V
the
DT
cake
with
the
fork

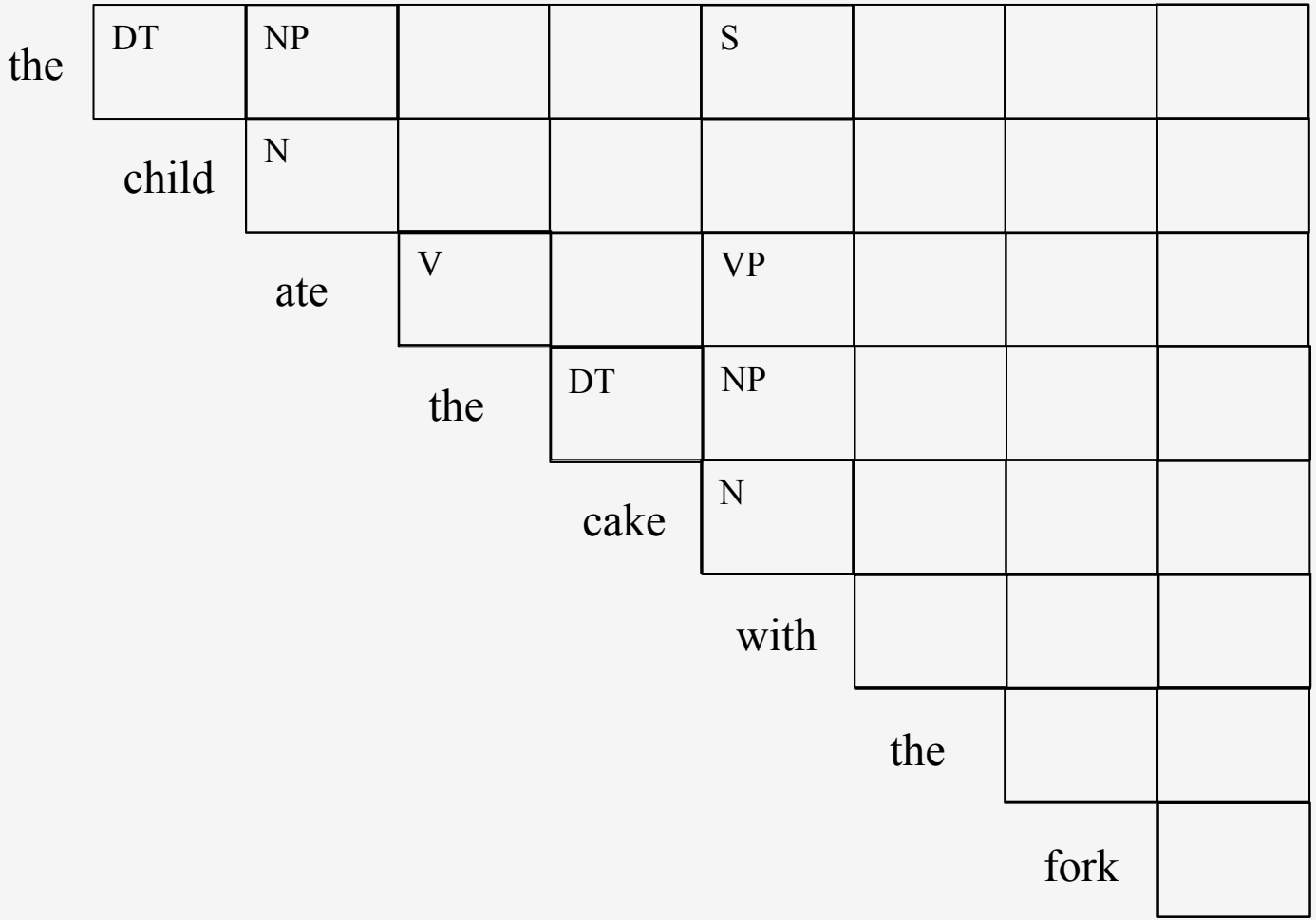

the
DT
NP
child
N
ate
V
the
DT
cake
N
with
the
fork

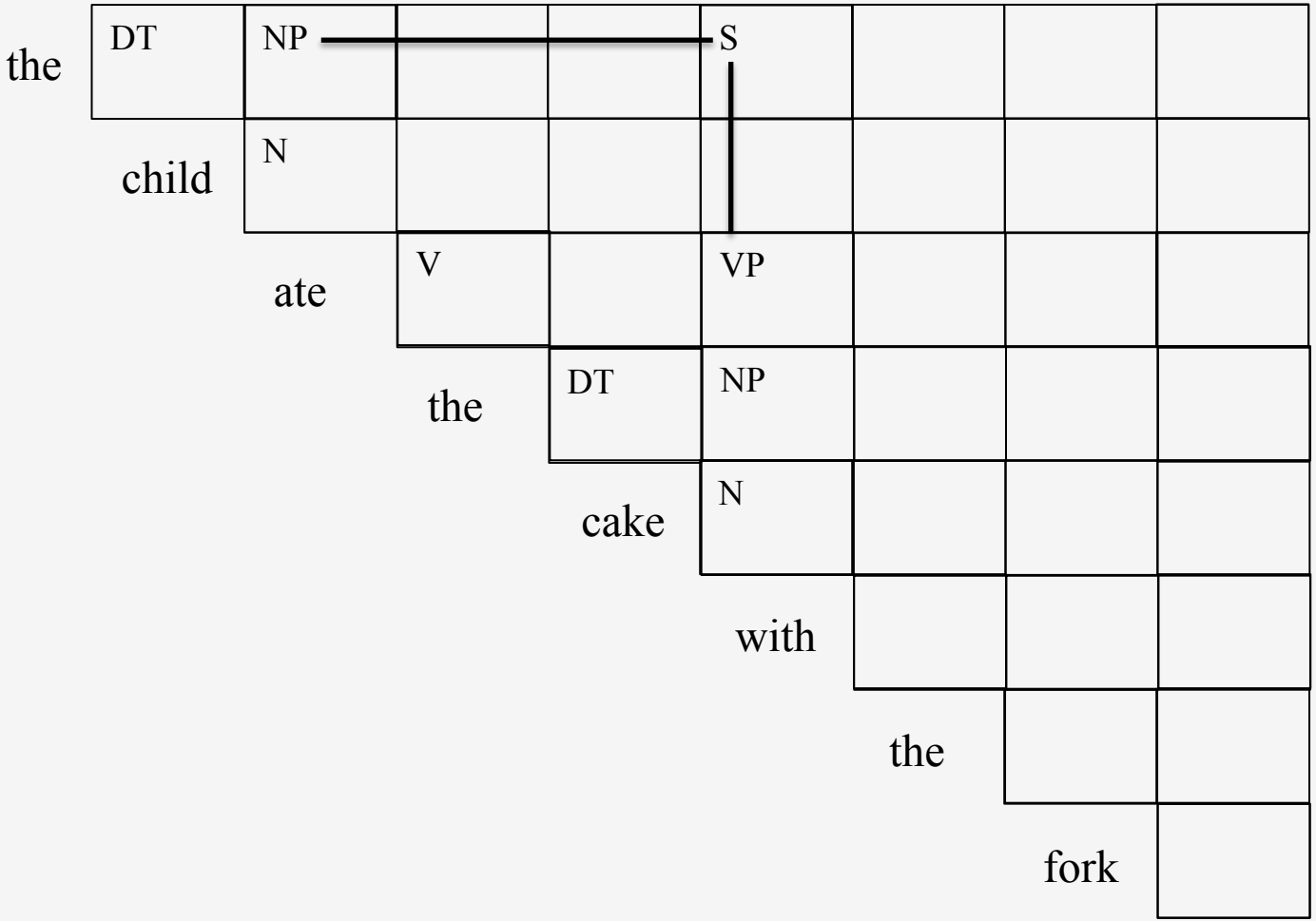

the
DT
NP
child
N
ate
V
the
DT
NP
cake
N
with
the
fork

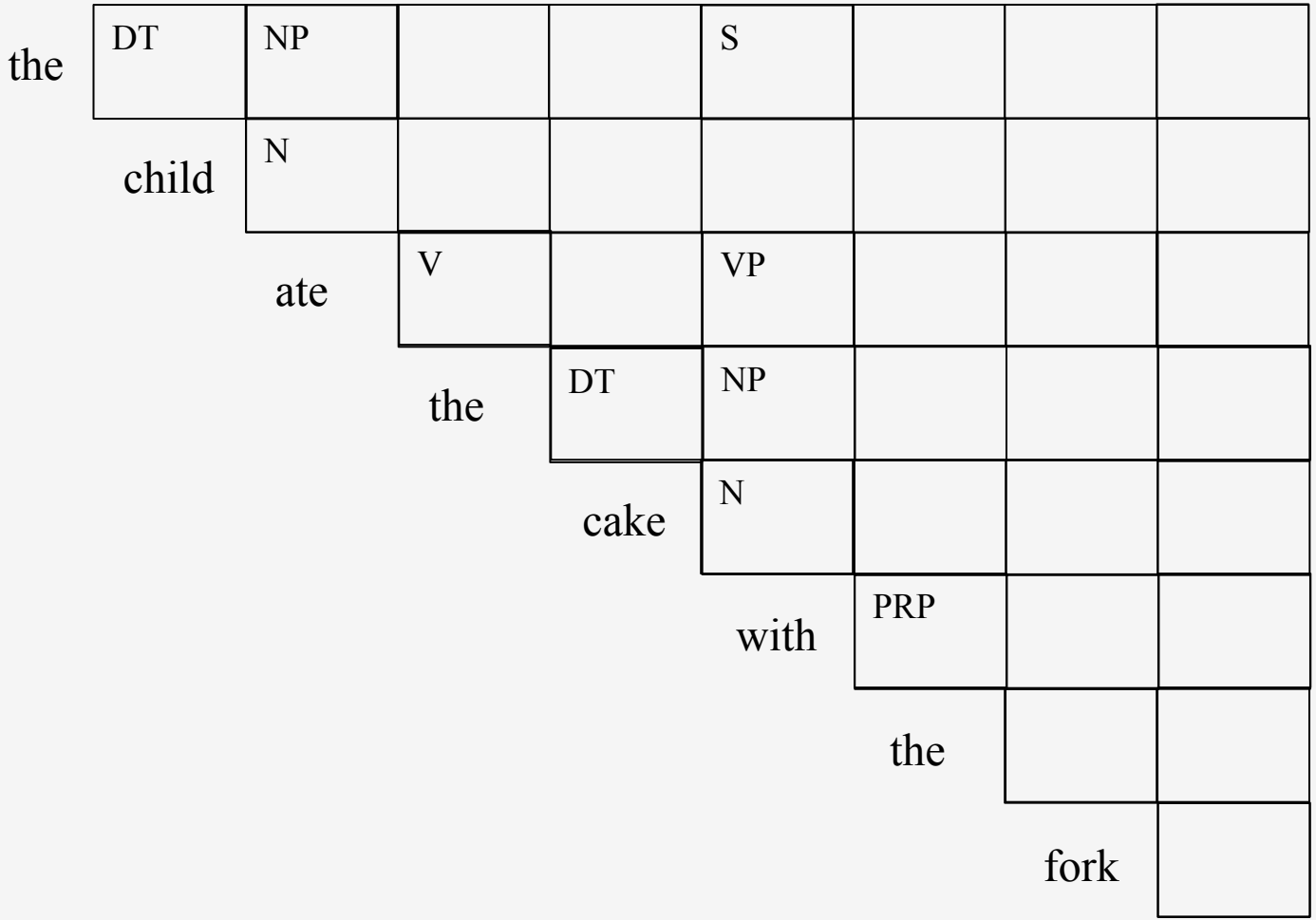

the
DT
NP
child
N
ate
V
the
DT
NP
N
cake
with
the
fork

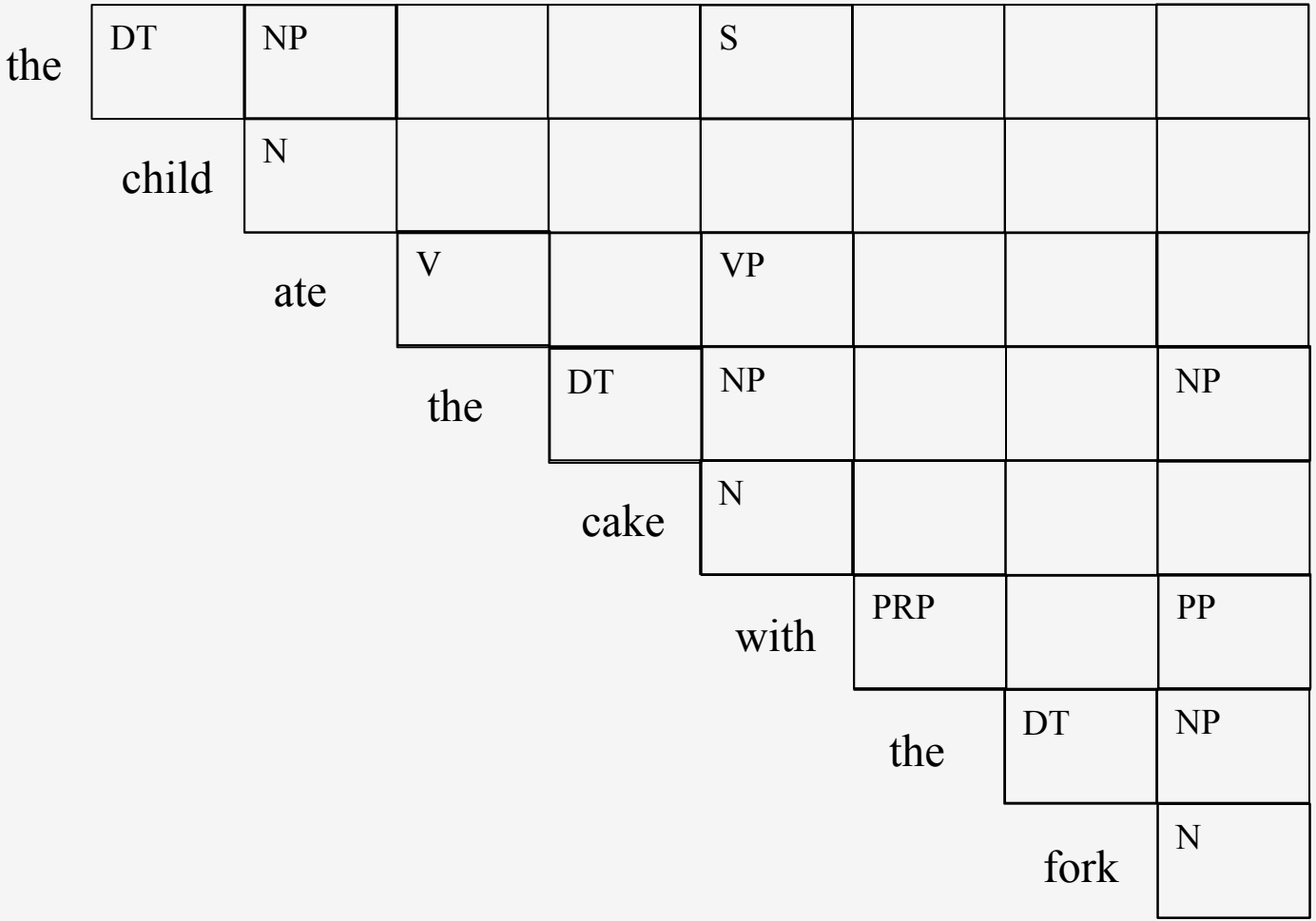

the
DT
NP
child
N
ate
V
VP
the
DT
NP
cake
N
with
the
fork

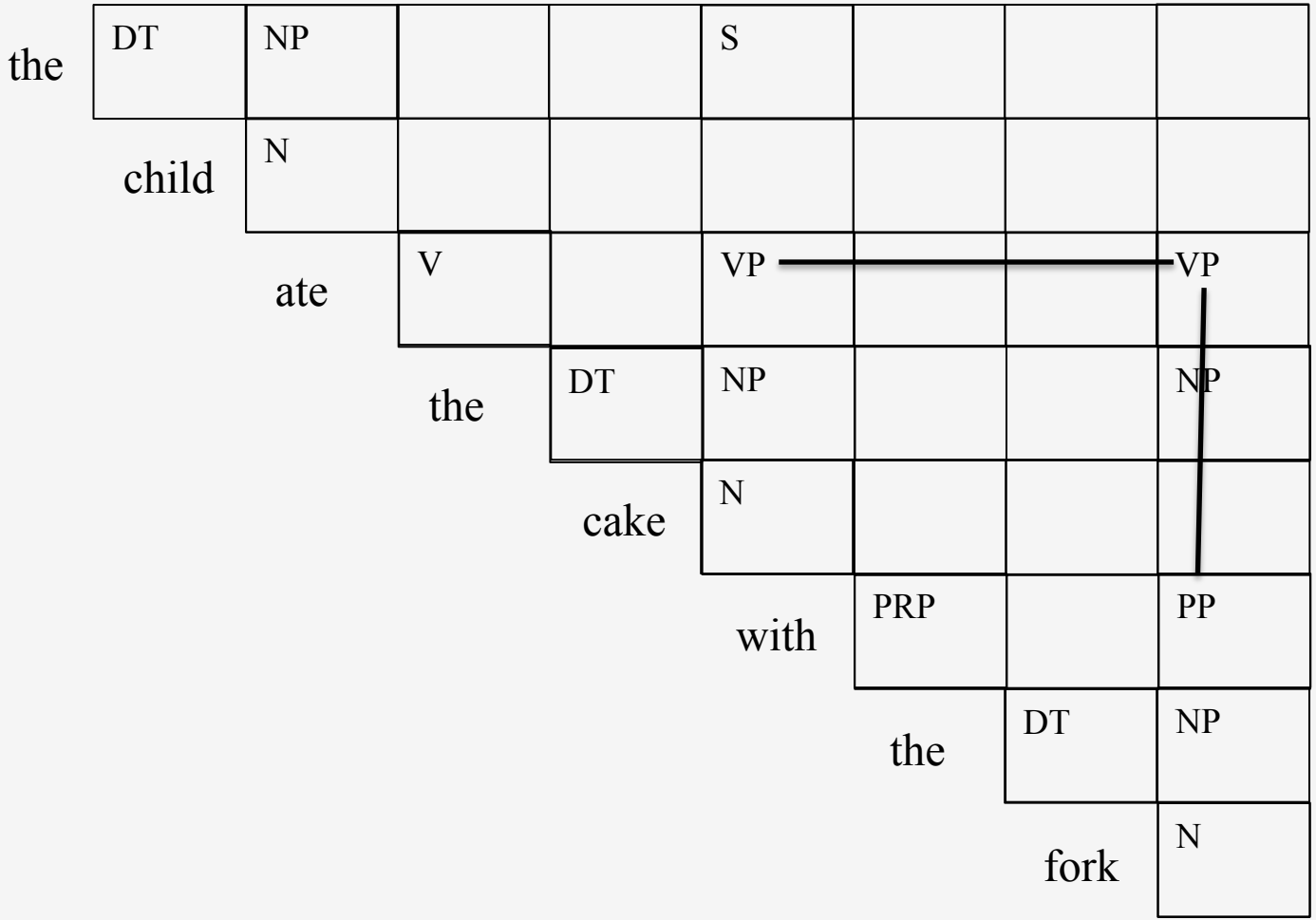

the
DT
NP
child
N
ate
V
VP
the
DT
NP
cake
N
with
the
fork

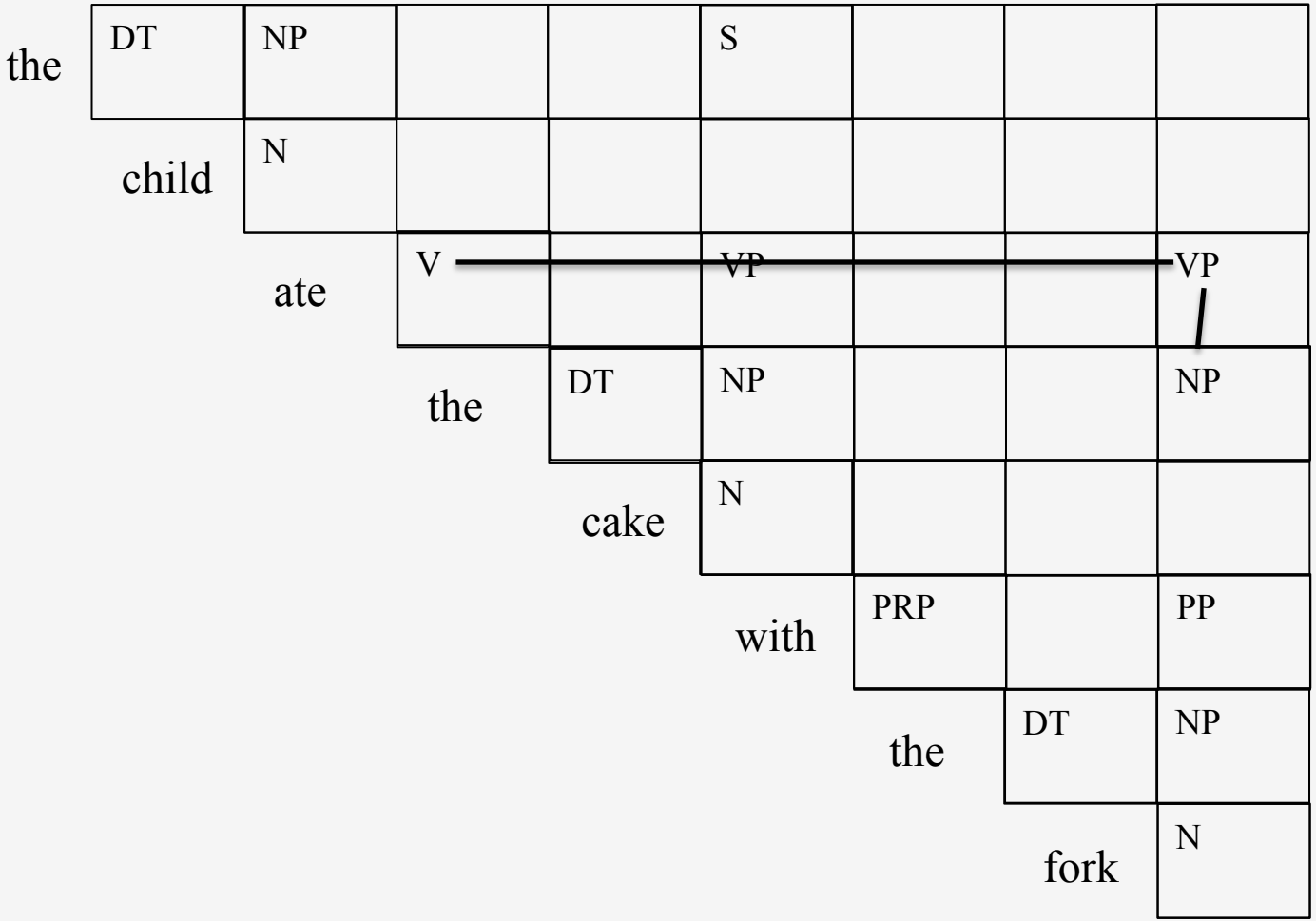

the
DT
NP
S
child
N
ate
V
VP
the
DT
NP
cake
N
with
the
fork

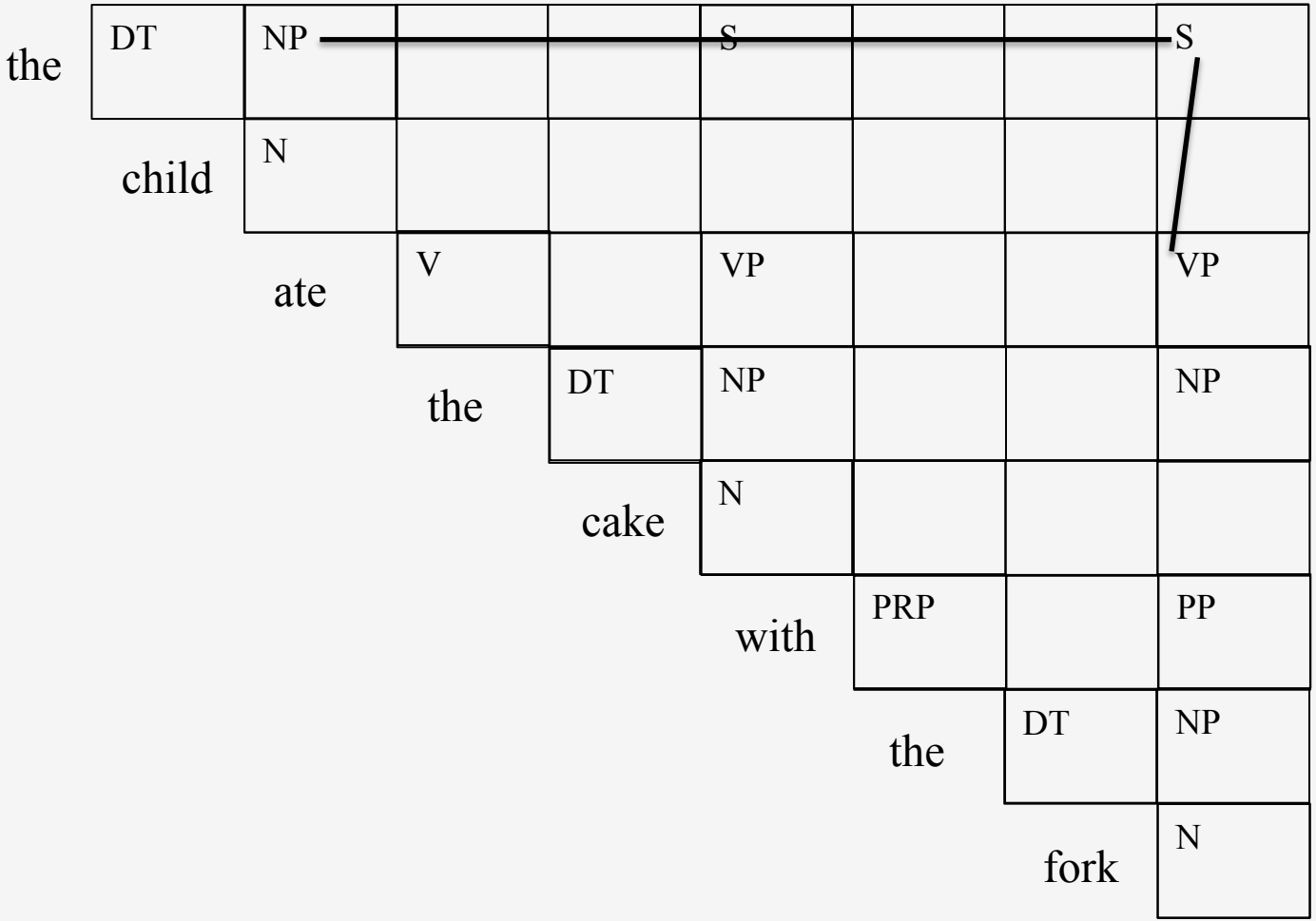

the
DT
NP
S
child
N
ate
V
VP
the
DT
NP
cake
N
with
the
fork

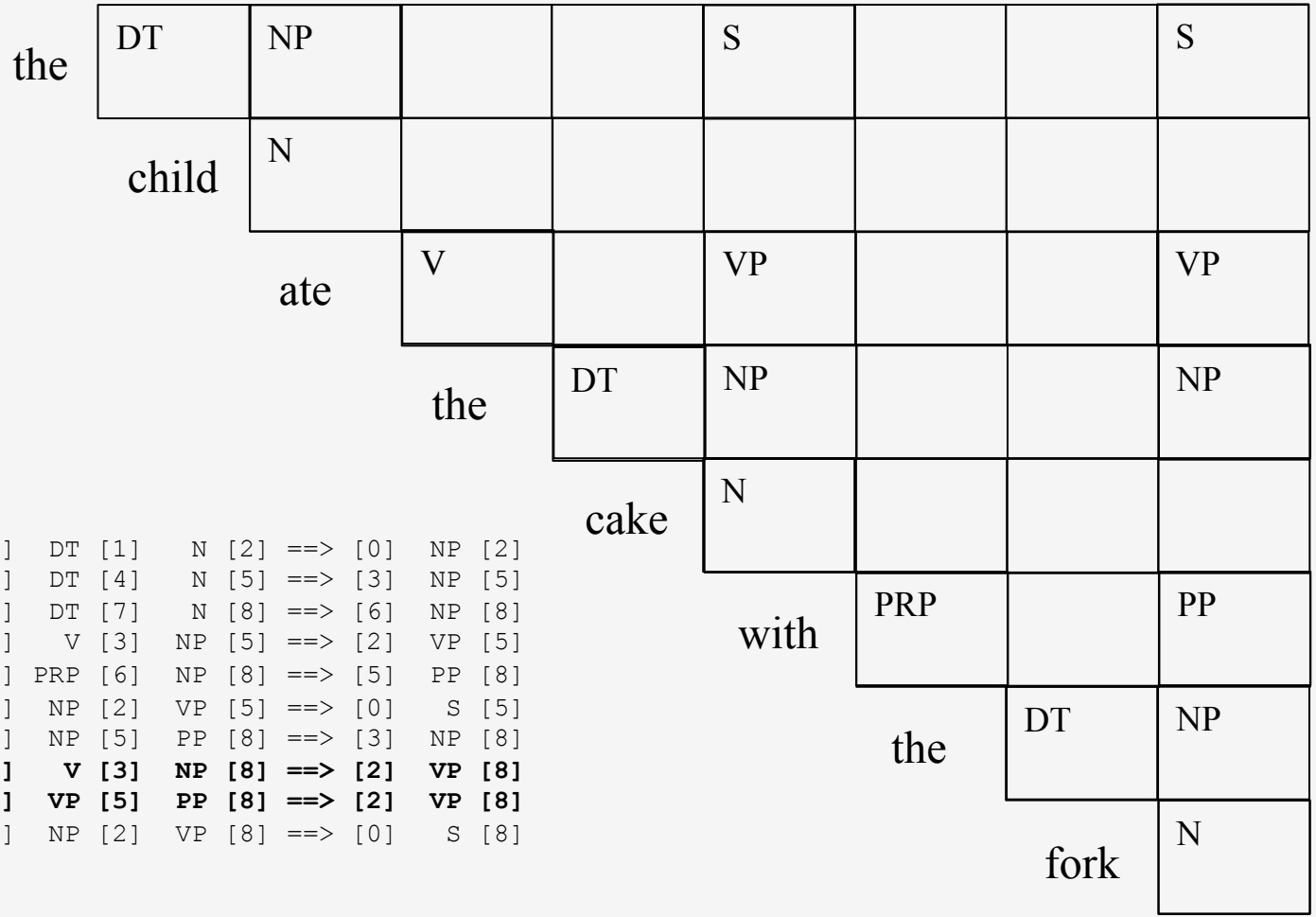

the
DT
NP
S
child
N
ate
V
VP
the
DT
NP
cake
N
with
PRP
the
fork


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

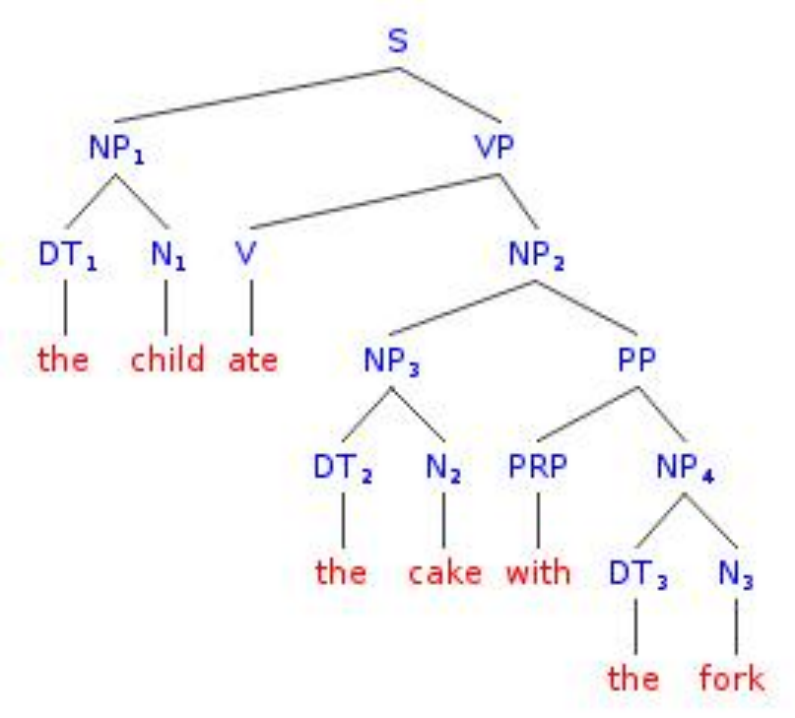

the
DT
NP
S
child
N
ate
V
VP
VP
the
DT
NP
NP
cake
N
with
PRP
PP
the
DT
NP
fork
N

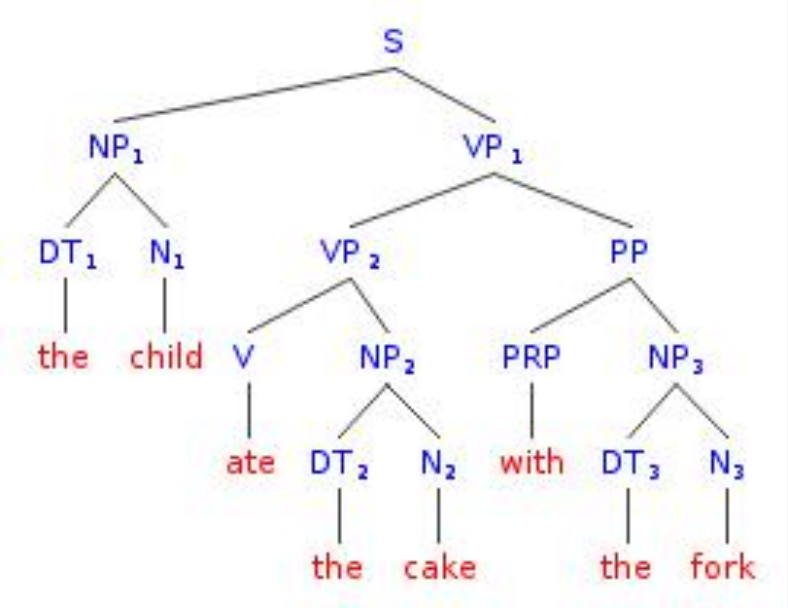

the
DT
NP
S
child
N
ate
V
VP
VP
the
DT
NP
NP
cake
N
with
PRP
PP
the
DT
NP
fork
N


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




```
[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))))))
```

Online Demo

- http://www.diotavelli.net/people/void/demos/cky.html

Complexity of CKY

- 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

- 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

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 → Y Z or X → w
- New non-terminals for hybrid rules, n-ary and unary rules:
  - INF-VP → to VP *becomes*
    - INF-VP → TO
    - TO → to
  - S → Aux NP VP *becomes*
    - S → R1 VP
    - R1 → Aux NP
  - S → VP VP → Verb VP → Verb NP VP → Verb PP *becomes*
    - S → book
    - S → buy
    - S → R2 PP
    - S → 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

NLP