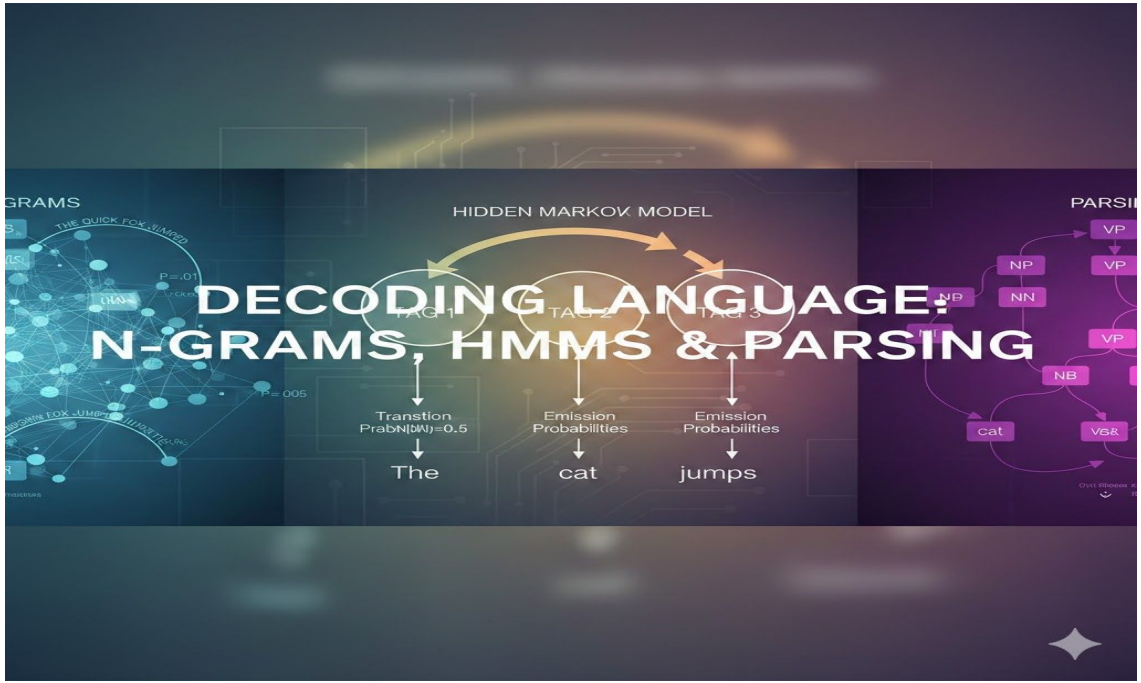


Natural Language Understanding

CSCI 4907/6515

Lecture 8: March 03, 2026

Aya Zirikly

Parsing

Parsing = Making explicit structure that is inherent (implicit) in natural language strings

Parsing = Making explicit structure that is inherent (implicit) in natural language strings

There are different ideas of what that structure is

Parsing can be useful for any application that needs access to the meaning of the text

Syntactic analysis can help NLP tasks by

I saw a girl with a telescope

Providing scaffolding for semantic analysis (and representing or resolving ambiguity)

Syntactic analysis can help NLP tasks by

I saw a girl with a telescope

After much economic progress over the years, the country has…

The country, which has made much economic progress over the years, still has…

Providing scaffolding for semantic analysis (and representing or resolving ambiguity)

Helping generalization (e.g., by capturing long-distance dependencies)

Constituency Parsing

Constituency Parsing

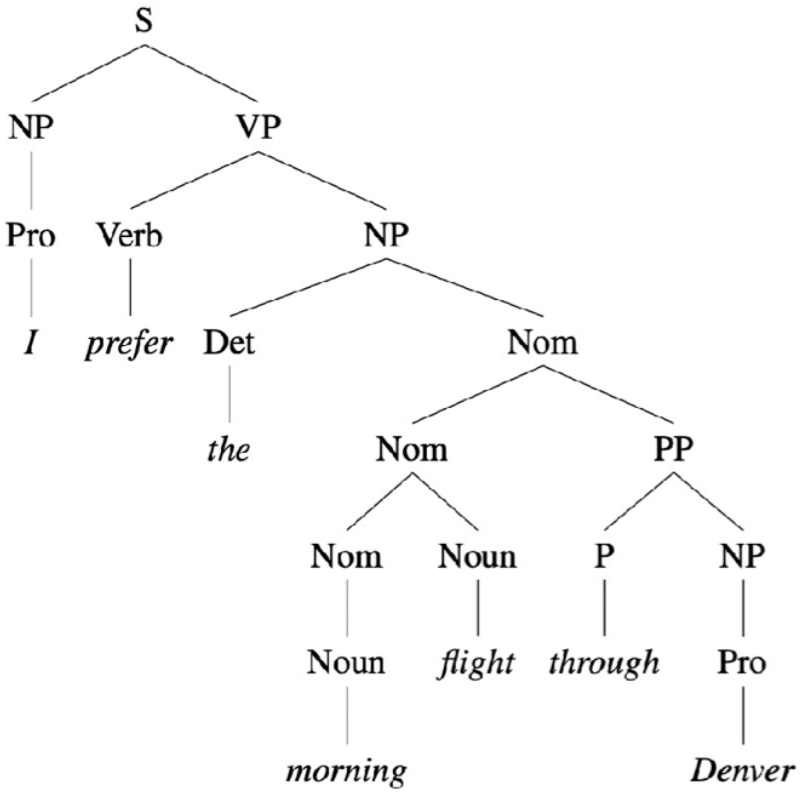
Constituency Parsing (also known as phrase structure parsing) is the process of analyzing a sentence by breaking it down into a hierarchy of sub-phrases, known as **constituents**.

A constituent is a group of words that functions as a single grammatical unit or block within a sentence. Instead of just looking at how individual words relate to one another, constituency parsing groups words into abstract grammatical categories, such as:

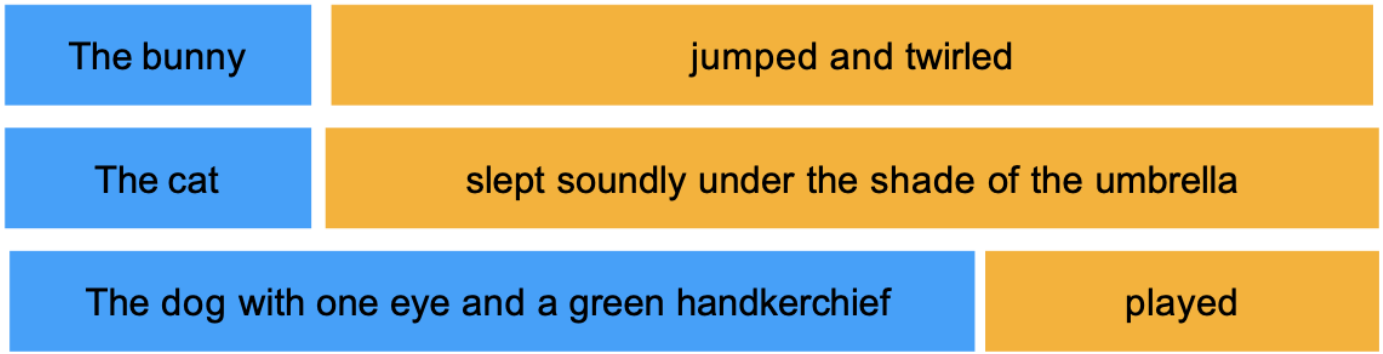
- **NP (Noun Phrase):** "The quick brown fox"
- **VP (Verb Phrase):** "jumped over the dog"
- **PP (Prepositional Phrase):** "over the dog"

Constituency Grammar

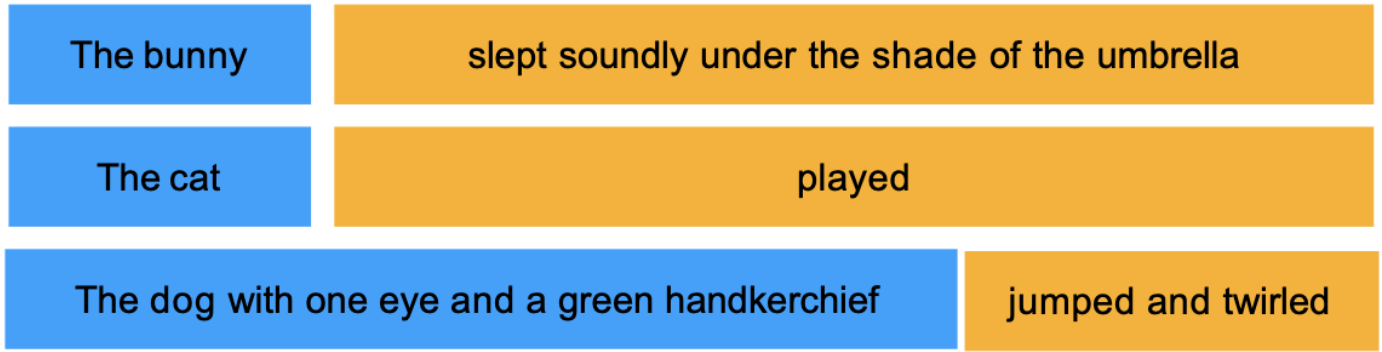
- AKA “Phrase Structure Grammar”
- Organize sentences into phrases that are functionally equivalent (w.r.t, the grammar)
- I.e., phrases can be swapped in and out and maintain grammaticality
  - (Note: grammatical != meaningful)



Constituency Grammar



Constituency Grammar



Context-Free Grammars = Phrase Structure Grammar

Consist of

- Set of rules or productions which dictate how the parts of the language can combine
  `S -> NP VP` (A Sentence is made of a Noun Phrase and a Verb Phrase).
- Lexicon (or vocabulary)
  - The dictionary that maps those abstract grammatical categories (Parts of Speech) to actual, tangible words.

```
Noun -> "dog" | "fox" | "elephant" | "pajamas"

Verb -> "jumped" | "shot" | "saw"

Determiner -> "the" | "a" | "an"

Preposition -> "over" | "in" | "with"
```

Constituent Structure: Equivalence Classes

Constituents act like swappable building blocks. Any phrase belonging to a specific category (like an NP) can be swapped with *any other phrase* in that same category without breaking the grammar.

The Substitution Test:

- **Phrase 1:** *"it"* (A simple 1-word Noun Phrase)
- **Phrase 2:** *"the lazy brown dog that the quick red fox jumped over"* (A complex 11-word Noun Phrase)

Because both are valid NPs, the grammar treats them identically. **Whenever *"it"* can appear in a sentence, the phrase "*the lazy brown dog that the quick red fox jumped over"* can also appear.**

Context-Free Grammar

- A CFG is a 4-tuple: < C, Σ, P, S >:
  - C is the set of *categories* (aka *non-terminals*, e.g., { S, NP, VP, V, ...})
  - Σ is the *vocabulary* (aka *terminals*, e.g., { Kim, snow, adores, ... })
  - P is the set of *rewrite rules*, of the form: α -> β1, β2, ..., βn
  - S (in C) is the *start-symbol*
  - For each rule α -> β1, β2, ..., βn in P, α is drawn from C and each β is drawn from C or Σ

$S \rightarrow NP\ VP$

$NP \rightarrow DET\ N$

$NP \rightarrow DET\ ADJ\ N$

$VP \rightarrow V_{intrans}$

$VP \rightarrow V_{trans}\ NP$

$DET \rightarrow the \mid a$

$N \rightarrow cat \mid dog \mid mouse$

$V_{intrans} \rightarrow slept \mid ate \mid ran$

$V_{trans} \rightarrow chased \mid bit$

$ADJ \rightarrow hungry \mid happy$

Grammar:

$S \rightarrow NP\ VP$

$NP \rightarrow DET\ N$

$NP \rightarrow DET\ ADJ\ N$

$VP \rightarrow V_{intrans}$

$VP \rightarrow V_{trans}\ NP$

Lexicon:

$DET \rightarrow the \mid a$

$N \rightarrow cat \mid dog \mid mouse$

$V_{intrans} \rightarrow slept \mid ate \mid ran$

$V_{trans} \rightarrow chased \mid bit$

$ADJ \rightarrow hungry \mid happy$

Where do these CFGs come from?

- Historically, linguists would write these rules out for languages by hand
- Data revolution: We can annotate existing data corpora and then extract all of the rules needed to generate the sentences in the corpus

Context-Free Grammars can be thought of as…

- A way of generating sentences
- A way of assigning structure to strings of words

Given a CFG, how do we generate a sentence?

Type	Rule
Rules	**S → NP VP**
	NP → Det N
	VP → V NP
Lexicon	**Det →** *the*
	N → *cat* | *mat*
	V → *sat on*

Strict notion of grammaticality

Grammatical = Generatable: If there is a path through your rules and lexicon that produces a specific sequence of words, that sentence is officially **grammatical**. Even if the sentence is nonsense (like "The colorless green ideas sleep furiously"), if the rules allow it, the grammar accepts it.

Ungrammatical = Impossible: If there is *no possible way* to combine your rules to form a specific sequence of words, that sentence is **ungrammatical**. This is the "strict" part: if the parser cannot find a valid tree, the sentence is rejected entirely.

Language = The Total Set: Under this definition, a "Language" is simply the collection of every single sentence that can be built using that grammar. If it's not in the set, it's not part of the language

In a strict grammar

- **No Room for Error:** A simple typo or a missing "the" would make a sentence "ungrammatical," and the parser would simply crash or return nothing.
- **No Ranking:** It can't tell you if one sentence is "more likely" than another; they are either 100% valid or 100% invalid.
- **Spoken Language:** Real people use fragments, slang, and messy structures that strict grammars can't handle.

Prerequisite: Chomsky Normal Form (CNF)

Definition: A restricted version of a CFG where all production rules must follow one of two patterns:

1. **Binary Branching:** A → B C (A non-terminal goes to exactly two non-terminals).
2. **Terminal Expansion:** A → w (A non-terminal goes to exactly one word).

This restriction ensures the parse tree is always binary, allowing the CKY table to combine exactly two sub-problems at every step.

Any Context-Free Grammar can be mathematically converted into an equivalent CNF grammar.

Given a CFG and a sentence, how do we get its parse(s)?

CYK (Cocke-Younger-Kasami) parsing

Grid representation

A table where the horizontal and vertical axes represent the positions between words in a sentence.

- **Indices:** If a sentence has n words, the grid is indexed from 0 to n.
- **Cells:** Each cell in the table (i, j) represents a "span" of words from position i to position j.
    - *Example:* In the sentence "Book that flight", the cell (0, 2) represents the span "Book that".

The Necessity of Binarization

The Problem: CKY requires Chomsky Normal Form (CNF).

Rules must be **Binary**: A → B C

Rules cannot be **n-ary**: S → NP VP PP (3 children)

The Solution: We use **Binarization** to break long rules into a sequence of pairs.

This creates **Intermediate Nodes** (e.g., X1, X2, X3) to hold partial structures.

Decoding S → X2 | PP

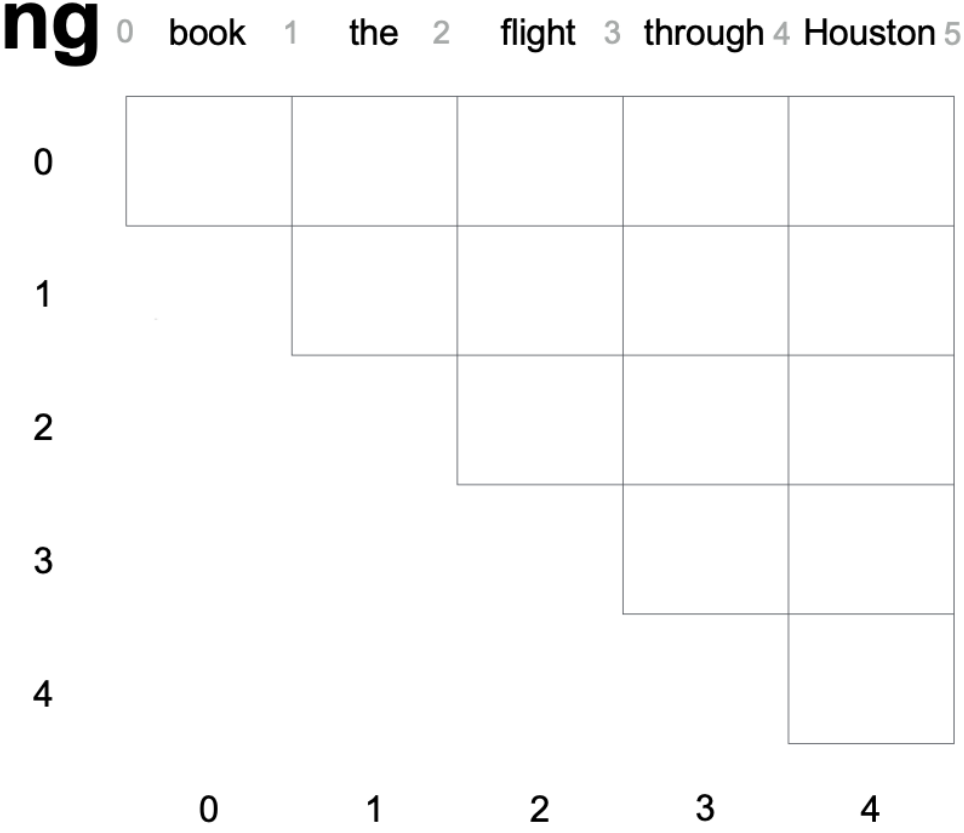
- **Original Rule:** S → NP VP PP
- **Binarized Version:**
  - X2 → NP | VP (The "X" is a temporary placeholder)
  - S → X2 | PP (The final "S" is completed)
- **What X2 represents:** X2 is an equivalence class for any sequence of a Noun Phrase followed by a Verb Phrase.
- **Trace in CKY:**
  - Find NP in Cell (i, k)
  - Find VP in Cell (k, l)
  - Write X2 in Cell (i, l)
  - Combine X2 with PP in the next step to reach S.

Summary

Cell Type	Coordinates	Meaning
Diagonal	(i, i+1)	Individual words (The "Lexicon" row).
Top Right	(0, n)	The entire sentence (The "Goal" cell).
Bottom Half	(j, i) where j > i	**Impossible/Invalid** (Backwards text).

Span Length	Cell to Fill	Combinations to Check	Split points (k)
1	(0, 1)	Lexicon for w_1	None
1	(1, 2)	Lexicon for w_2	None
1	(2, 3)	Lexicon for w_3	None
2	(0, 2)	(0, 1) + (1, 2)	k = 1
2	(1, 3)	(1, 2) + (2, 3)	k = 2
3	(0, 3)	**(0, 1) + (1, 3)** OR **(0, 2) + (2, 3)**	k = 1, k = 2

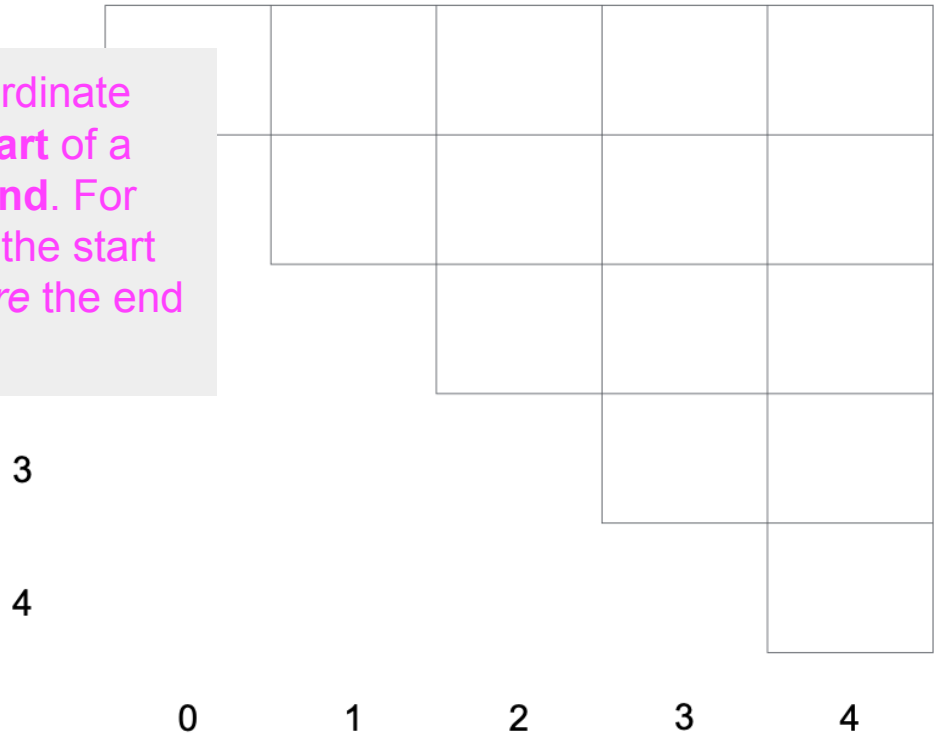
CKY Parsing



CKY Parsing

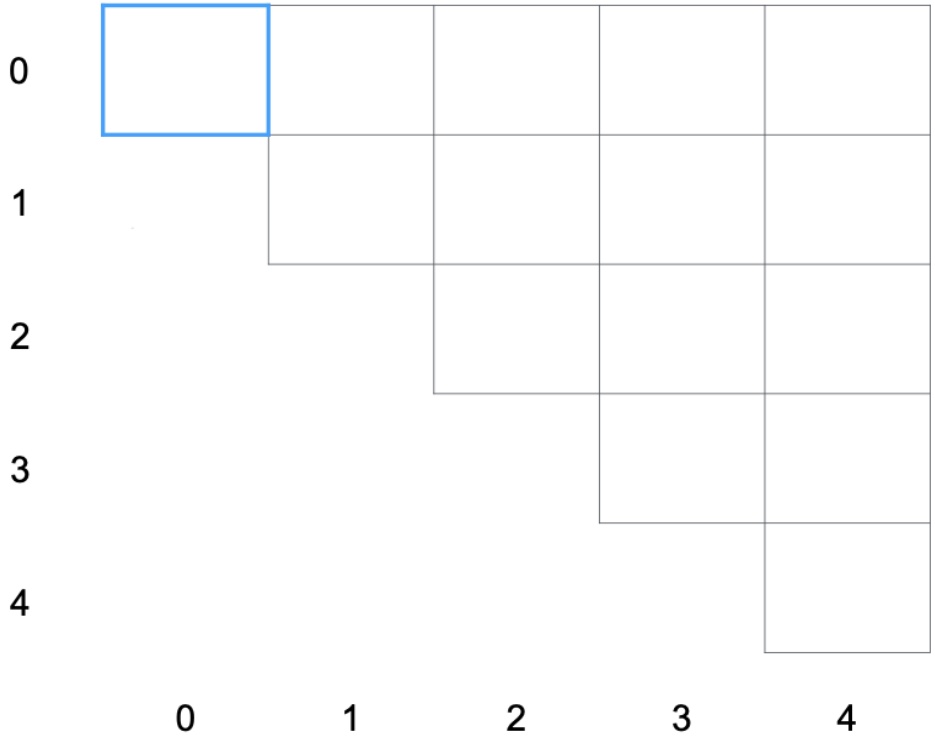
0 book 1 the 2 flight 3 through 4 Houston 5

In CKY, the grid is a coordinate system where i is the **start** of a word span and j is the **end**. For the grid to make sense, the start must always come *before* the end (i < j).



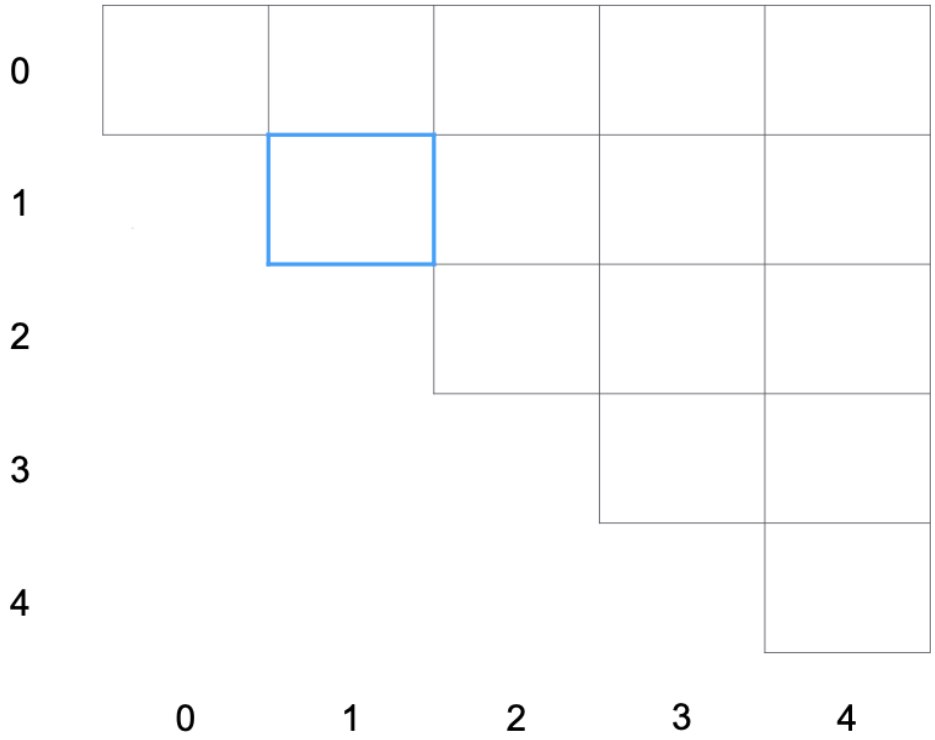
CKY Parsing

0 **book** 1 the 2 flight 3 through 4 Houston 5



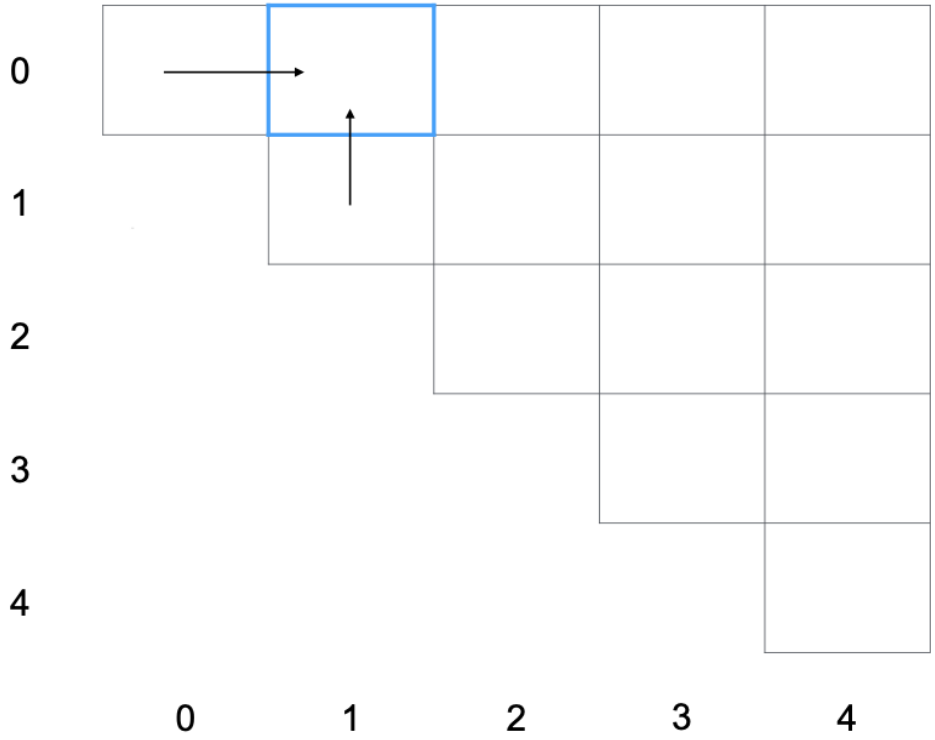
CKY Parsing

0 book 1 the 2 flight 3 through 4 Houston 5

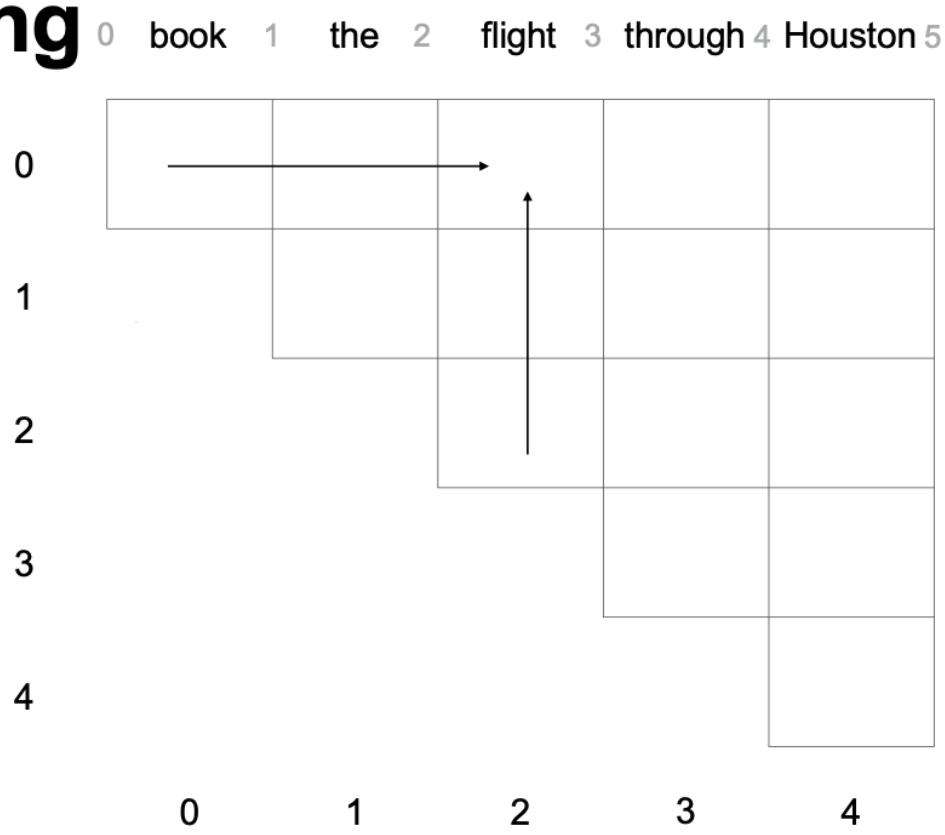


CKY Parsing

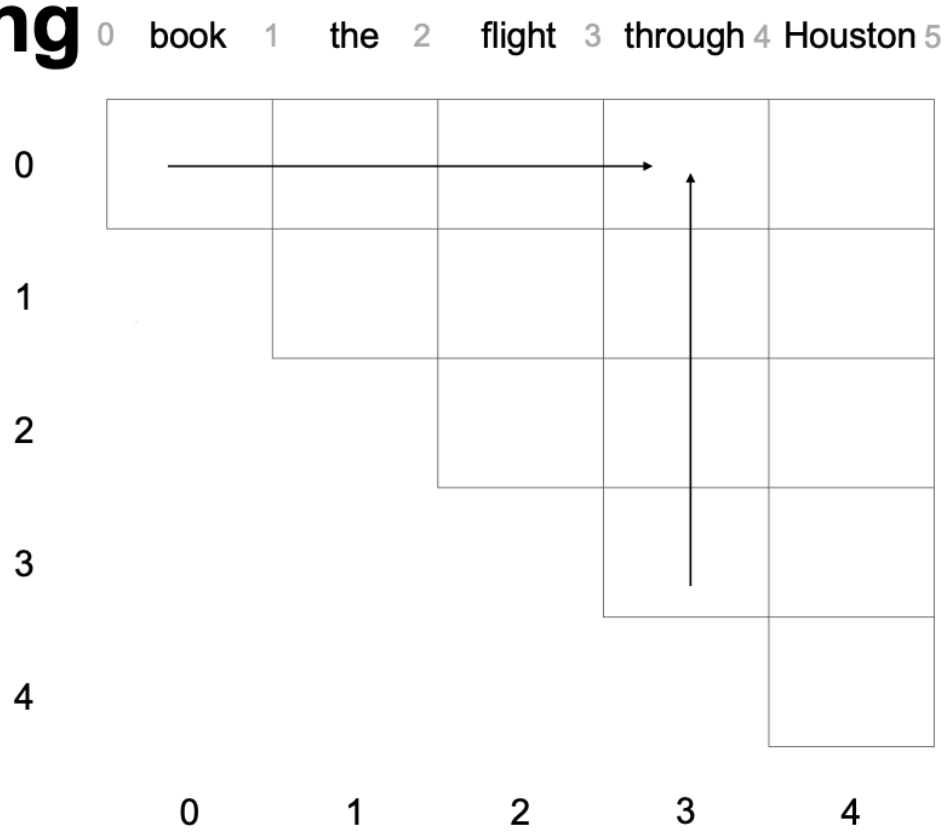
0 book 1 the 2 flight 3 through 4 Houston 5



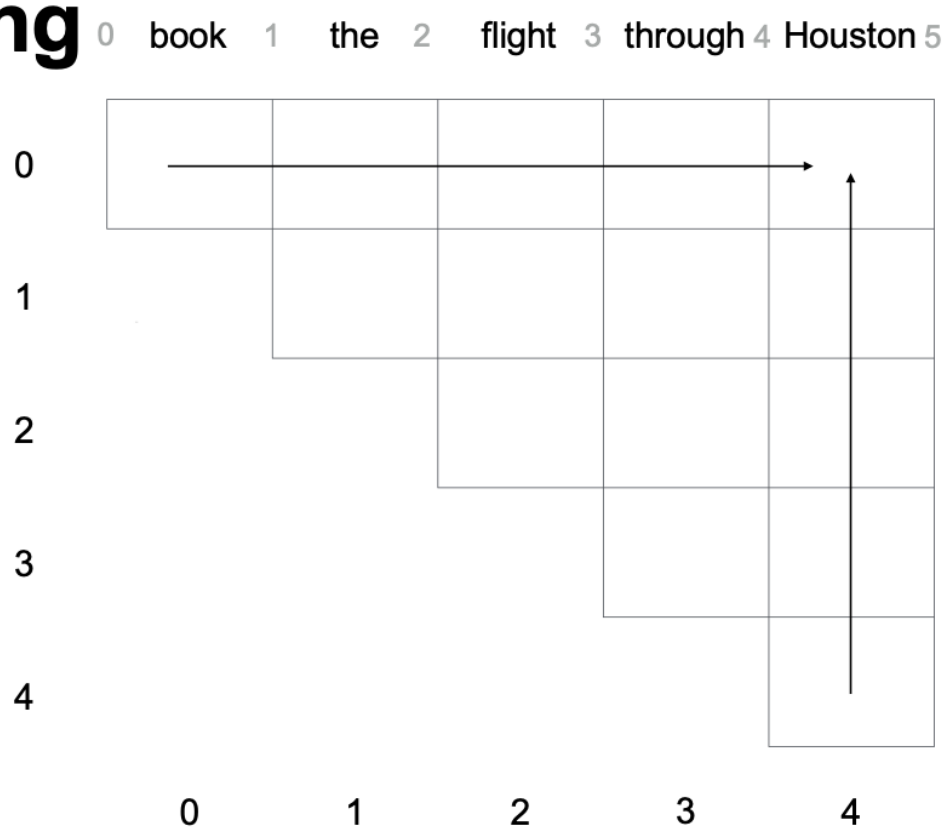
CKY Parsing



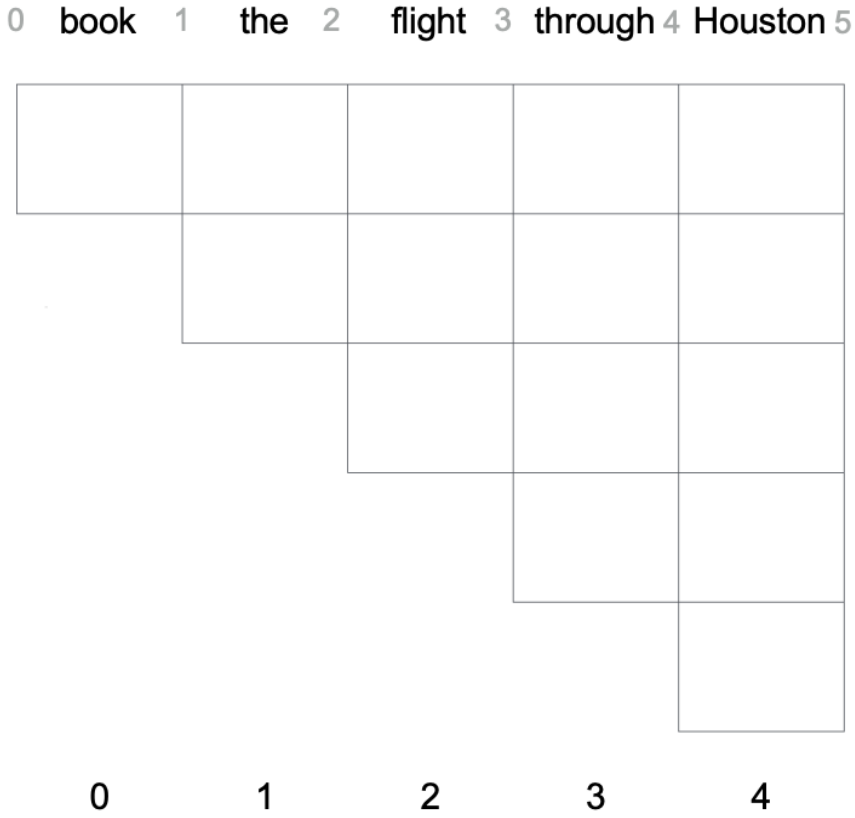
CKY Parsing



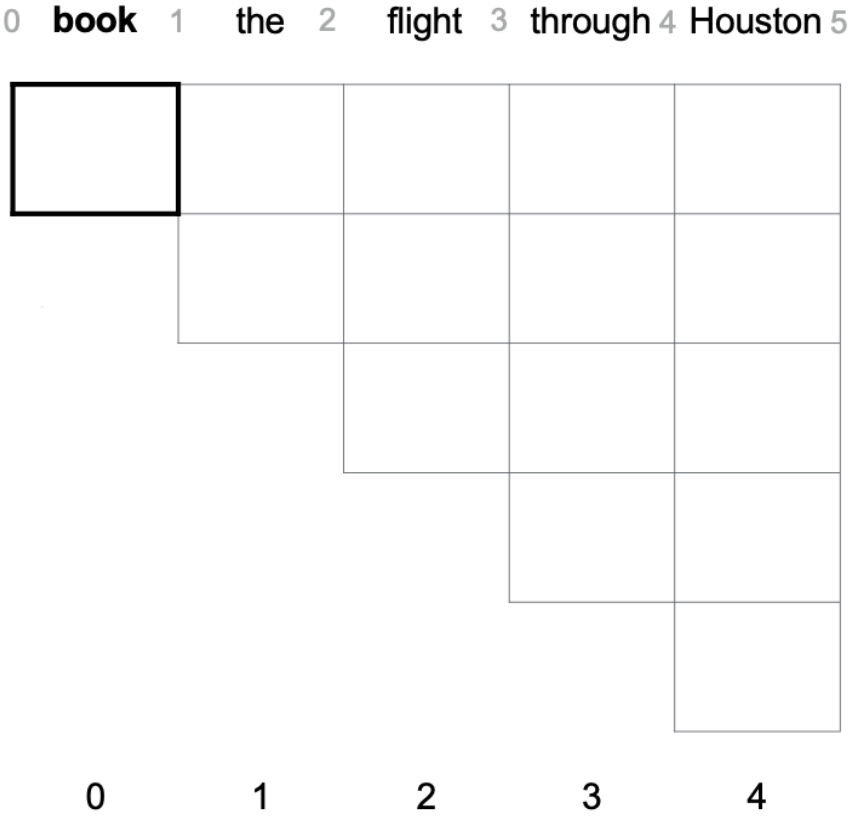
CKY Parsing



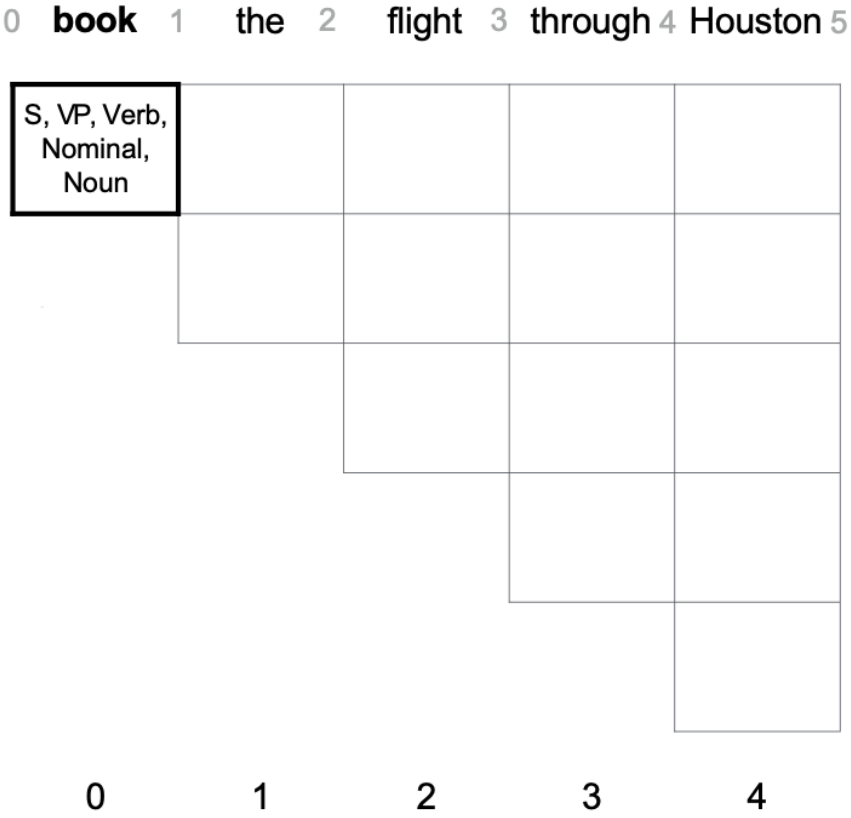
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | this | a | the
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



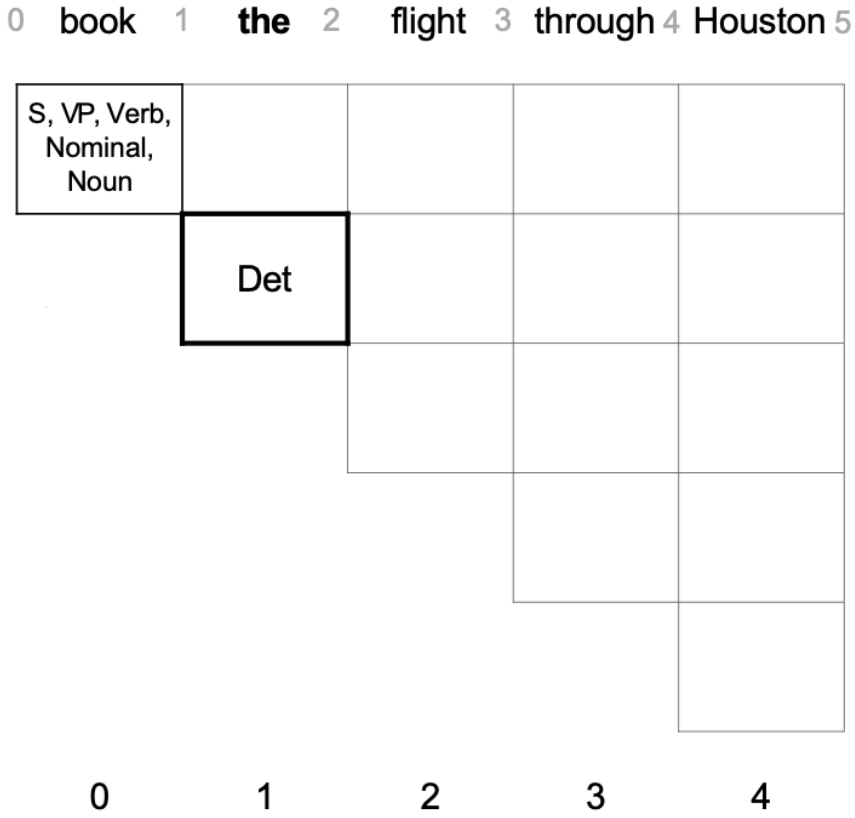
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | this | a | the
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



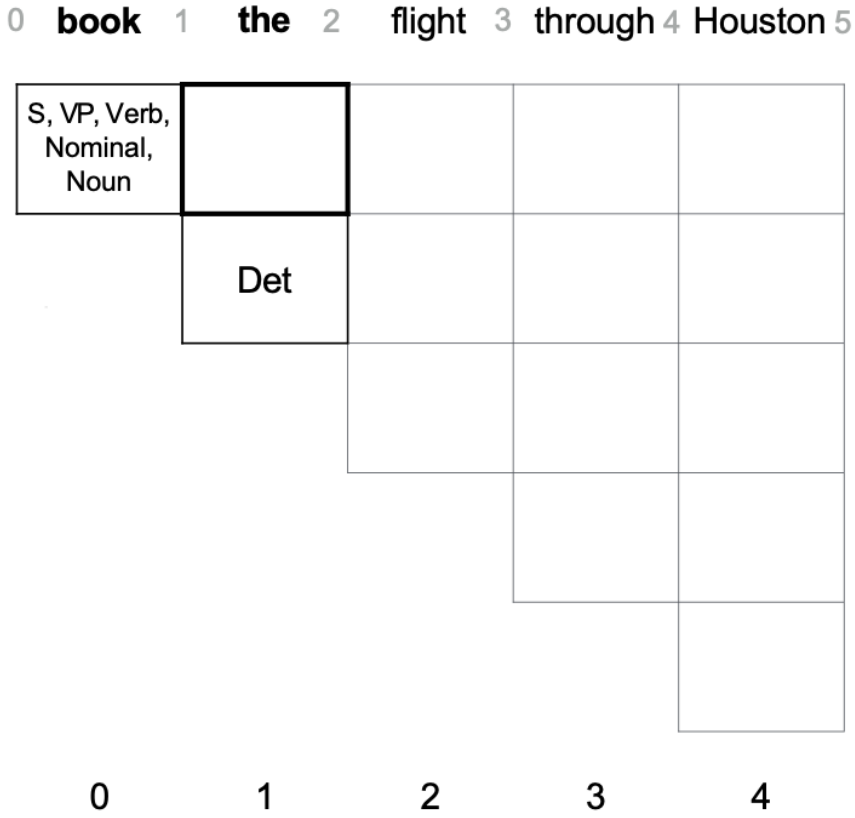
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | this | a | the
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



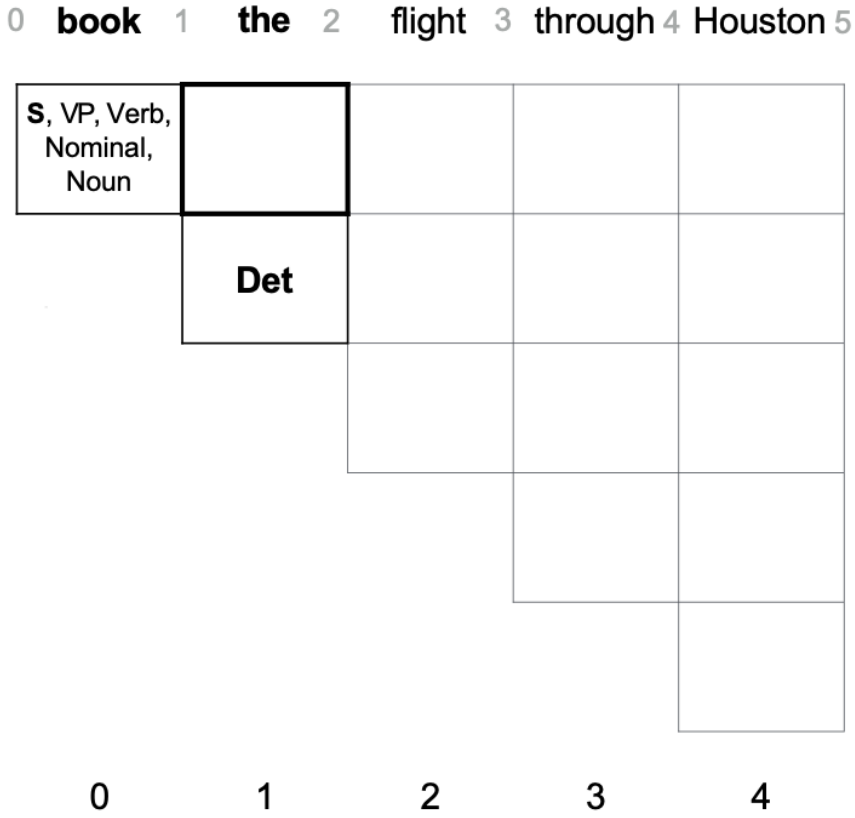
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | **the** | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



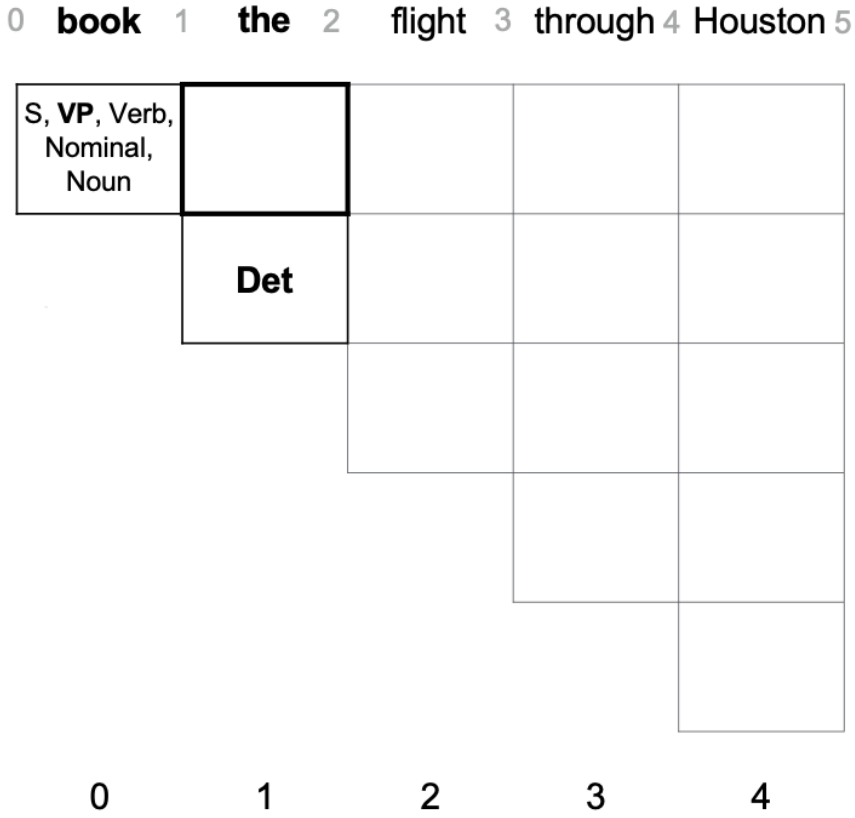
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



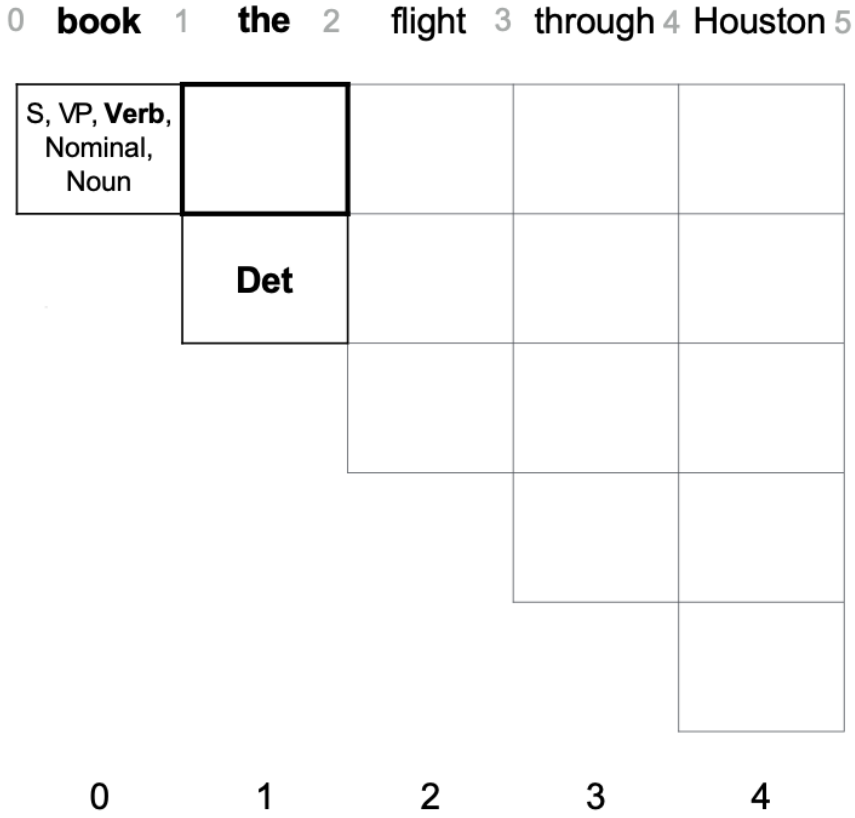
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

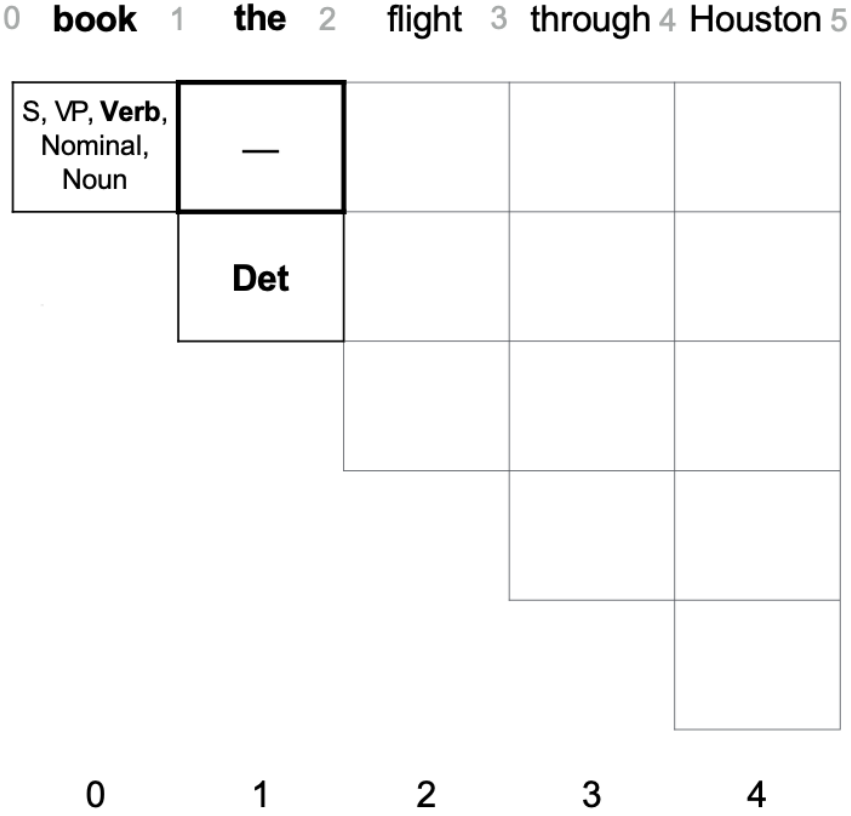


S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

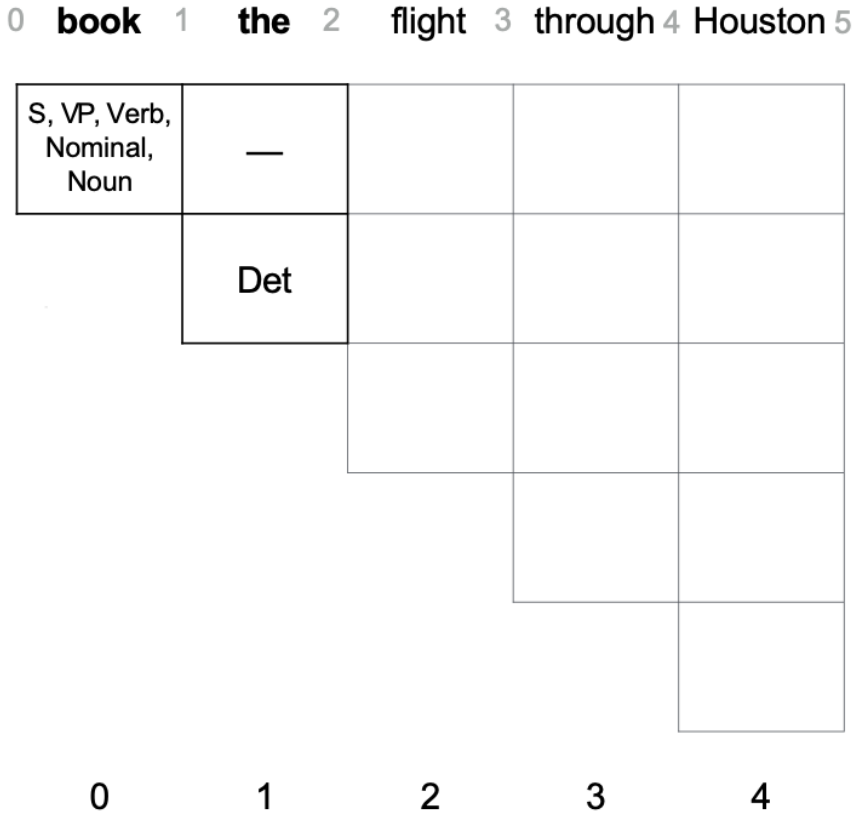


S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 1 2 3 4

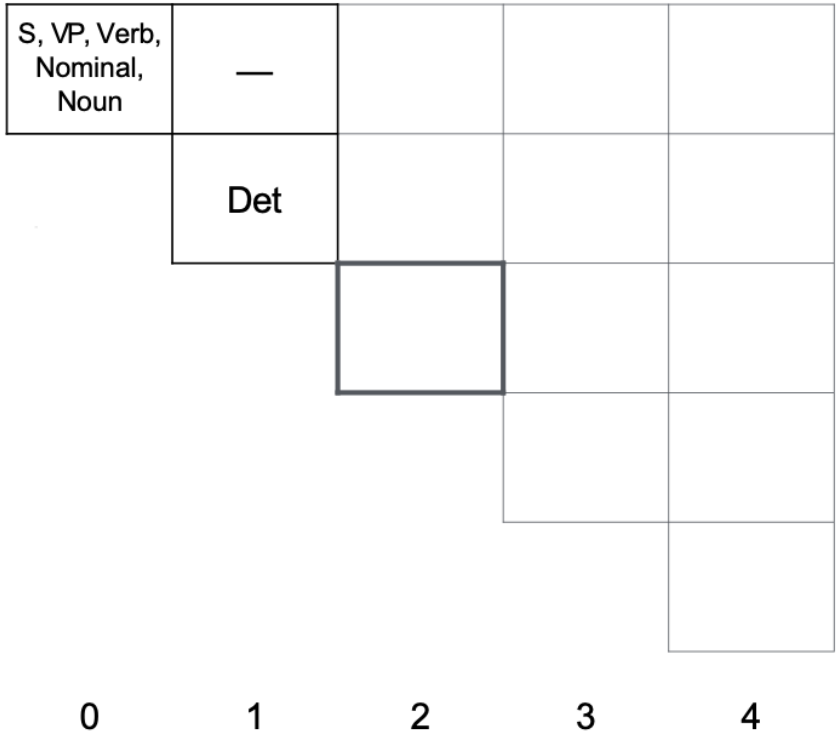


S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

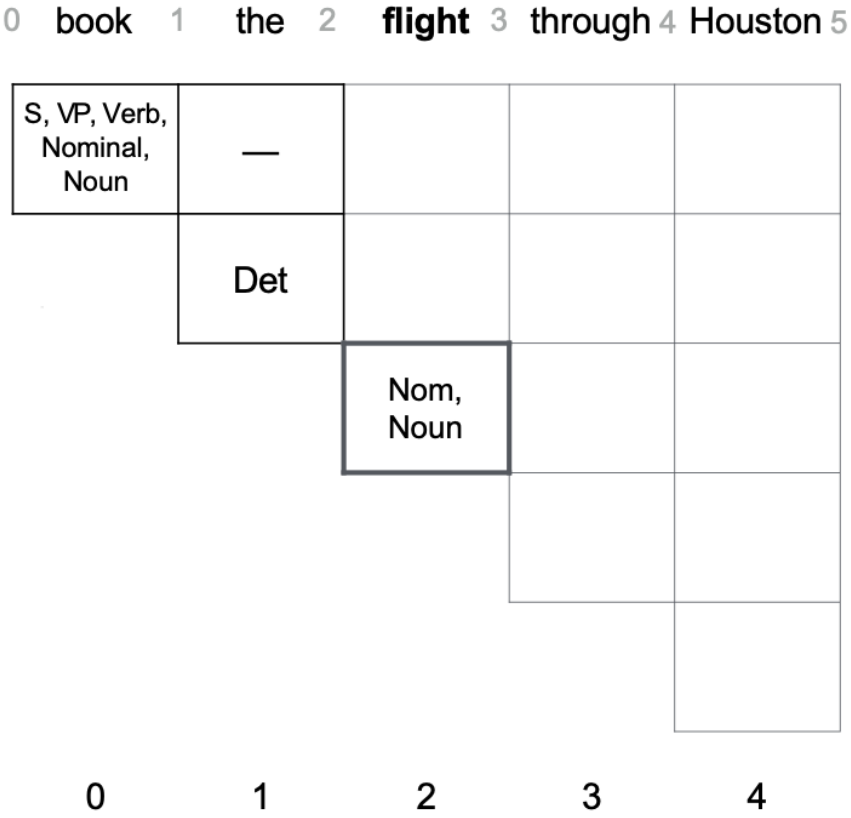


S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

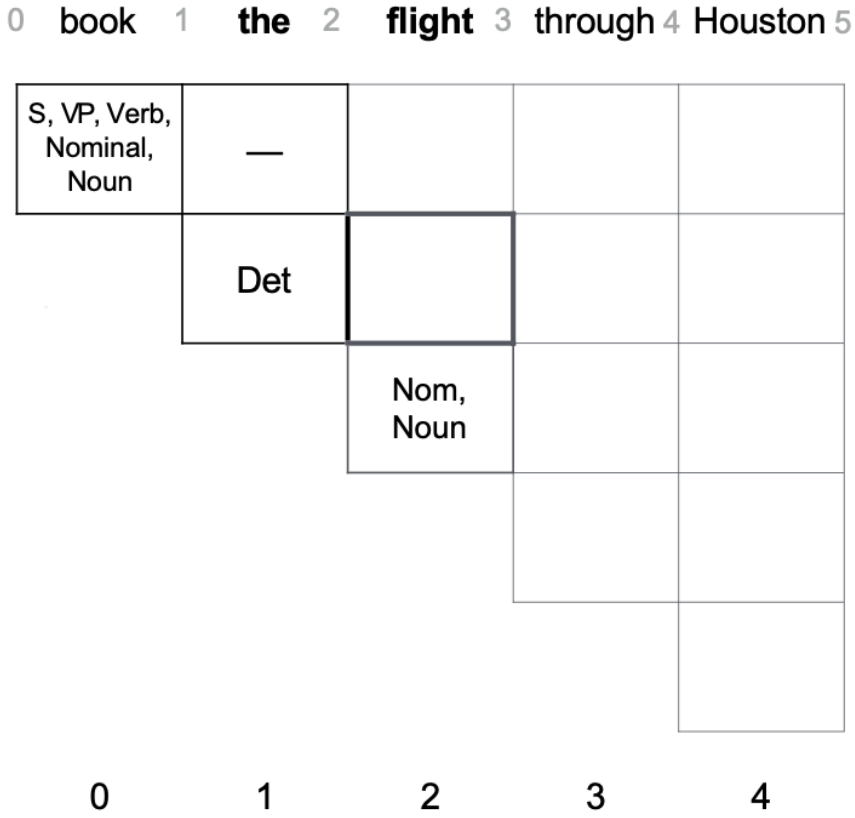
0 book 1 the 2 **flight** 3 through 4 Houston 5



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | **flight**
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | **flight**
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



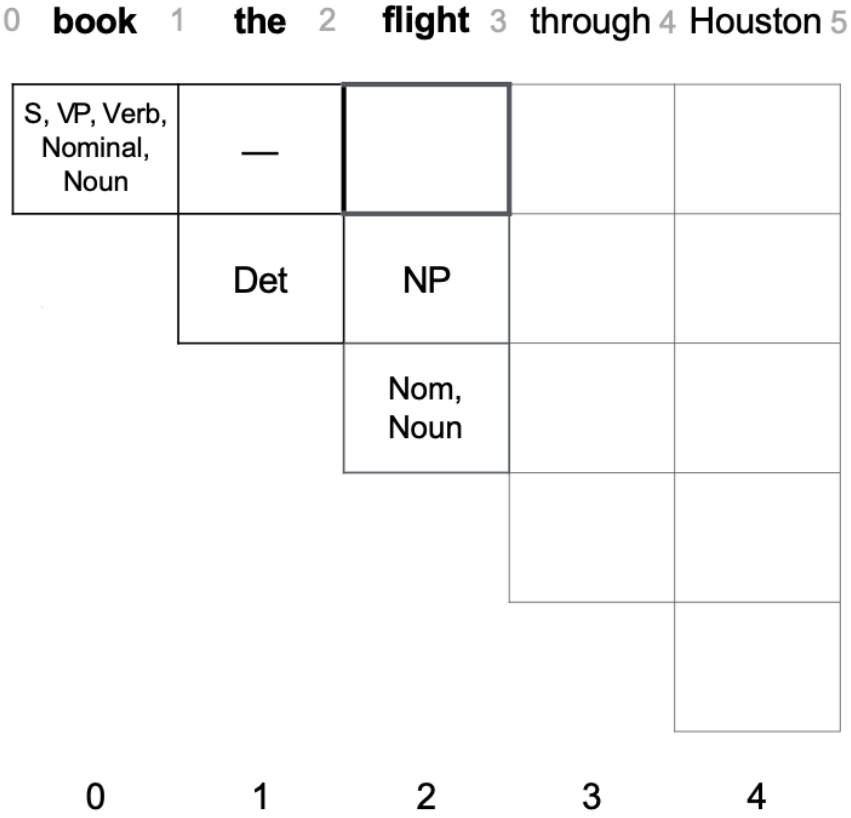
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 book 1 **the** 2 **flight** 3 through 4 Houston 5

S, VP, Verb, Nominal, Noun	—			
	Det	NP		
		Nom, Noun		

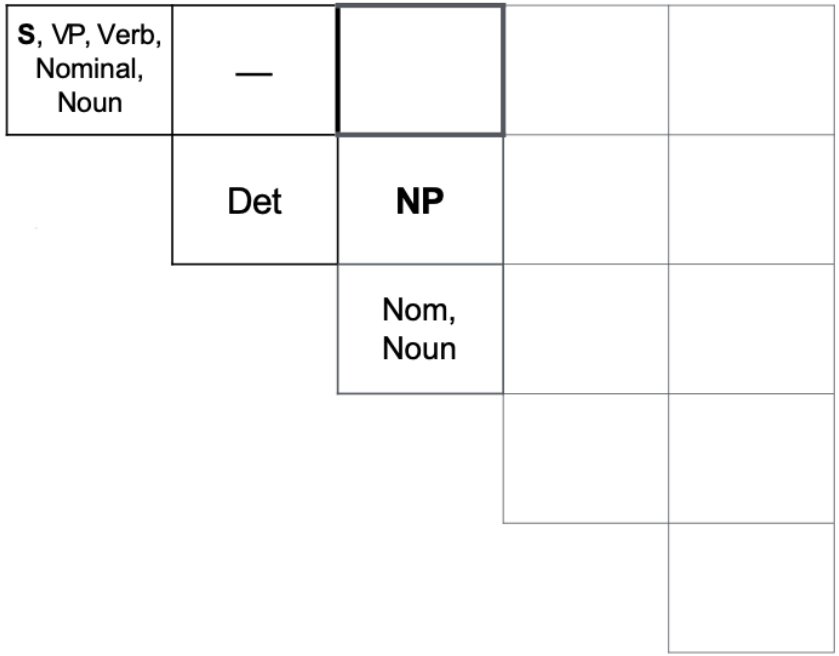
0 1 2 3 4

S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 **book** 1 **the** 2 **flight** 3 through 4 Houston 5



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

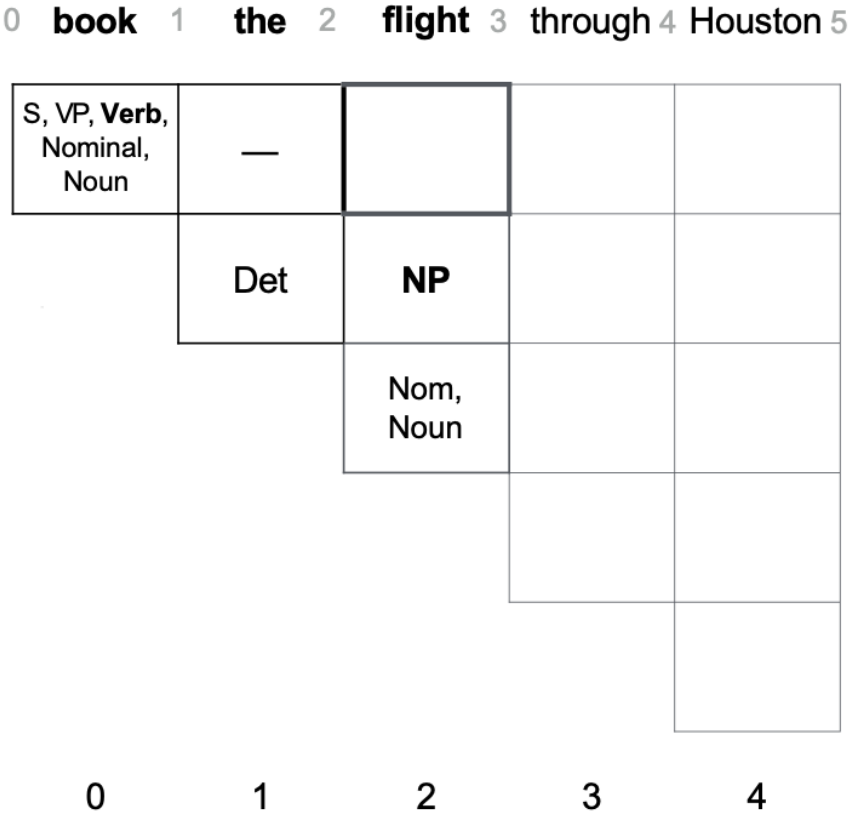
0 1 2 3 4

0 **book** 1 **the** 2 **flight** 3 through 4 Houston 5

S, **VP**, Verb, Nominal, Noun	—			
	Det	**NP**		
		Nom, Noun		

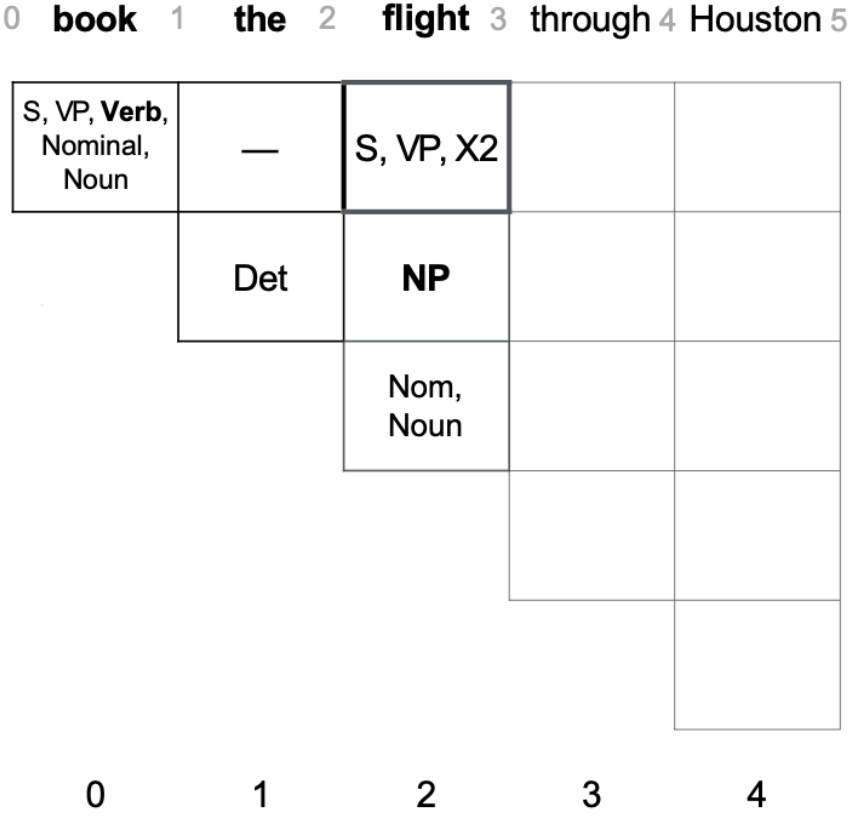
0 1 2 3 4

S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 1 2 3 4



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 book 1 the 2 flight 3 through 4 Houston 5

S, VP, Verb, Nominal, Noun	—	S, VP, X2		S, VP
	Det	NP		NP
		Nom, Noun		Nom
			Prep	PP
				NP, Prop-N

0 1 2 3 4

S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
Prep. → from | to | through

0 book 1 the 2 flight 3 through 4 Houston 5

S, VP, Verb, Nominal, Noun	—	S, VP, X2		S, VP
	Det	NP		NP
		Nom, Noun		Nom
				PP
				NP, Prop-N

0 1 2 3 4

Left Cell (0,3)	Right Cell (3,5)	Rule Triggered	Added to Cell (0,5)
X2	**PP**	VP → X2 PP S → X2 PP	**VP, S**
VP	**PP**	VP → VP PP	**VP**

S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

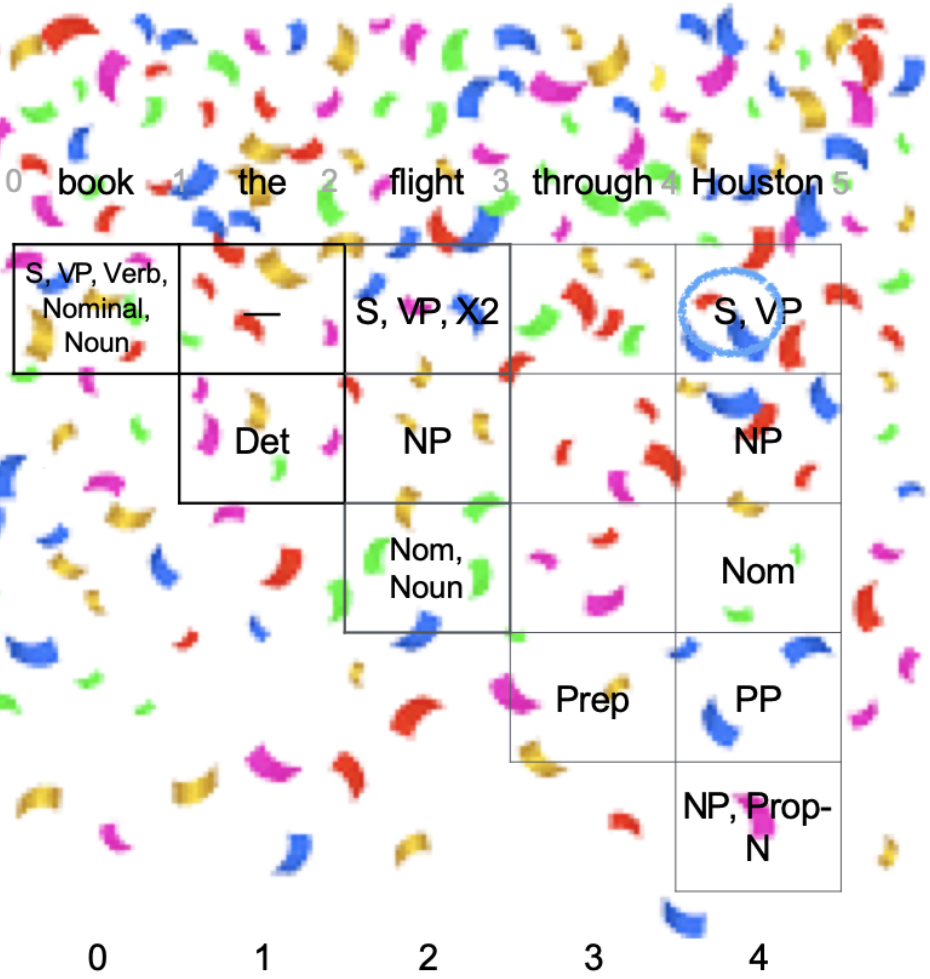
0 1 2 3 4

0 book 1 the 2 flight 3 through 4 Houston 5

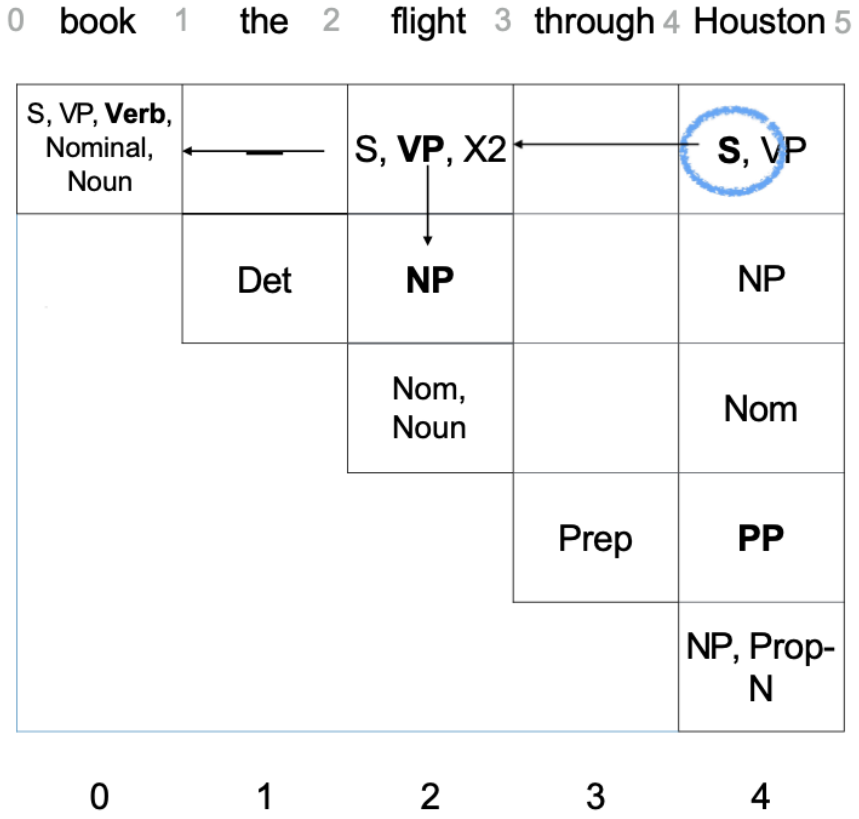
S, VP, Verb, Nominal, Noun	—	S, VP, X2		S, VP
	Det	NP		NP
		Nom, Noun		Nom
			Prep	PP
				NP, Prop-N

0 1 2 3 4

S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through



Question: Can we build more than one tree for a sentence? How does CKY handle ambiguity?

Can we build more than one tree?

Structural Ambiguity: Most natural language sentences have multiple valid grammatical interpretations.
The "Telescope" Example: "I saw the man with the telescope."

- **Interpretation 1 (VP-attachment):** I used a telescope to see the man. (The telescope modifies the action "saw").
- **Interpretation 2 (NP-attachment):** I saw a man who happened to be carrying a telescope. (The telescope modifies the "man").

CKY's Role: The algorithm is **exhaustive**. It does not "choose" one tree; it finds and stores **every** valid path allowed by the grammar.
The Result: One sentence = One CKY table = A "Forest" of possible trees.

Mechanical Handling of Ambiguity

Multiple Labels: If a span of words (like "the man with the telescope") can be both a simple NP or a complex NP, CKY stores **both** versions in cell (2, 7).
Back-Pointers: Each label in a cell acts as a "pointer" to the two cells below it that created it.
Overlapping Paths: In the top-right cell (0, 7), the "S" might have two different ways to form:

- **Path A:** NP ("I") + VP ("saw the man with the telescope")
- **Path B:** NP ("I") + VP ("saw the man") + PP ("with the telescope")

Efficiency: CKY saves space by "sharing" sub-trees. The NP "the man" is calculated **once**, even if it appears in five different potential trees.

But, Context Free Grammars don’t allow us to prefer some parses over others

Disambiguation with Probabilistic CKY

The Problem: CKY finds all trees, but it doesn't know which one is "correct" or most likely in the real world.
The Solution: Probabilistic Context-Free Grammars (**PCFG**).
Process:

1. Every rule is assigned a probability (e.g., P(VP → VP PP) = 0.3 or P(NP → NP PP) = 0.6).
2. As cells are filled, CKY calculates a **Score** for each label.
3. **Formula:** Score = P(Rule) * P(Left Child) * P(Right Child).

Selection: At the end, we follow the back-pointers of the **S** in cell (0, n) that has the **highest total probability**. This is the "Viterbi" parse.

Probabilistic Context-Free Grammars (PCFG)

- A CFG is a 4-tuple: < C, Σ, P, S >:
  - C is the set of *categories* (aka *non-terminals*, e.g., { S, NP, VP, V, ...})
  - Σ is the *vocabulary* (aka *terminals*, e.g., { Kim, snow, adores, ... })
  - P is the set of *rewrite rules*, of the form: α -> β1, β2, ..., βn each with a probability p of β1, β2, ..., βn given α
  - S (in C) is the *start-symbol*
  - For each rule α -> β1, β2, ..., βn in P, α is drawn from C and each β is drawn from C or Σ

PCFG

$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow kids$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow birds$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow saw$	0.04
$P \rightarrow with$	1.0	$NP \rightarrow fish$	0.18
$V \rightarrow saw$	1.0	$NP \rightarrow binoculars$	0.1

Probabilities of rewrite rules for a particular non-terminal need to sum to 1

Given a PCFG, how do we generate sentences?

$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow$ kids	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow$ birds	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow$ saw	0.04
$P \rightarrow$ with	1.0	$NP \rightarrow$ fish	0.18
$V \rightarrow$ saw	1.0	$NP \rightarrow$ binoculars	0.1

Given a PCFG & sentence, how to get parse(s)?

Probabilistic CKY Parsing: store parses with probabilities

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
N 1.0, NP 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06		S 0.2 · 0.0036 · 0.8	John
	V 1.0, VP 0.3	VP 1 · 0.3 · 0.2 = 0.06		VP max(1.0 · 0.008 · 0.3, 0.06 · 0.2 · 0.3)	eats
		N 1.0, NP 0.2		NP 0.2 · 0.2 · 0.2 = 0.008	pie
			P 1.0	PP 1 · 1 · 0.2	with
				N 1.0, NP 0.2	cream

```
S    → NP VP          0.8
S    → S conj S       0.2
NP   → Noun           0.2
NP   → Det Noun       0.4
NP   → NP PP          0.2
NP   → NP conj NP     0.2
VP   → Verb           0.3
VP   → Verb NP        0.3
VP   → Verb NP NP     0.1
VP   → VP PP          0.3
PP   → P NP           1.0
```

Probabilities in PCFGs

- The probability of a tree is a product of the probabilities of the rules used to generate it
- The probability of a sentence is the sum of all possible trees that generate it (language model)

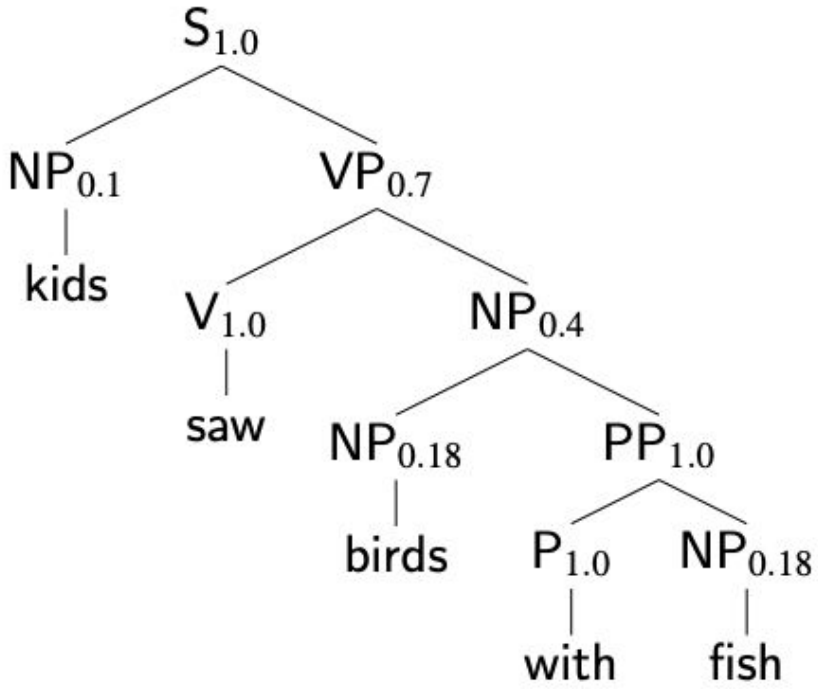
Concept	Action	Logic
Tree Probability	**Multiply**	Every rule in the tree must happen to create that specific structure.
Sentence Probability	**Sum**	The sentence exists if *any* of its valid trees exist.

This allows us to prefer parses over others…

Parse: Kids saw birds with fish

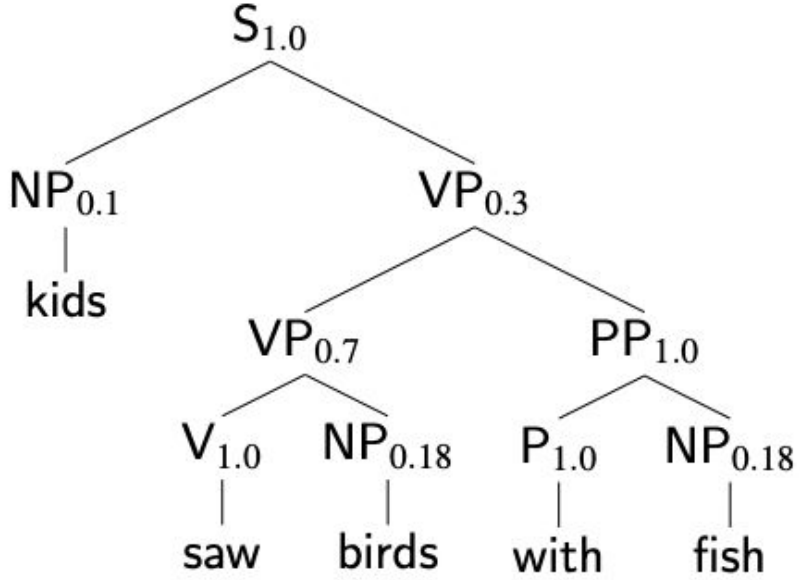
$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow$ kids	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow$ birds	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow$ saw	0.04
$P \rightarrow$ with	1.0	$NP \rightarrow$ fish	0.18
$V \rightarrow$ saw	1.0	$NP \rightarrow$ binoculars	0.1

Probability of parse 1



- $P(t_1) = 1.0 \cdot 0.1 \cdot 0.7 \cdot 1.0 \cdot 0.4 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0009072$

Probability of parse 2



- $P(t_2) = 1.0 \cdot 0.1 \cdot 0.3 \cdot 0.7 \cdot 1.0 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0006804$
- which is less than $P(t_1) = 0.0009072$, so t_1 is preferred. Yay!

Does this solve the issue of ambiguity?

Does this solve the issue of ambiguity?

No! Expansion of a node does not depend on context

No Context

It means the grammar rules are "blind." A node does not know who its neighbors are, and it doesn't care.

- If you have the rule V → ate, the parser will put "ate" under a Verb node whether the Subject is "The cat" or "The pizza."
- **The Problem:** In real world, "The pizza ate the cat" is grammatically valid but logically weird. A CFG doesn't care about the meaning or the surrounding words; it only cares that a Verb follows a Noun Phrase.

But there are still problems with ambiguity

Replacing one word with another with the same POS will never result in a different parsing decision, even though it should!

- kids saw birds with fish vs.
  kids saw birds with binoculars

- She stood by the door covered in tears vs.
  She stood by the door covered in ivy

- stray cats and dogs vs.
  Siamese cats and dogs

How to fix PCFGs?

Lexicalization (The "Head Word" Fix)

Instead of just having a category like **VP**, we annotate every node with its **Head Word** (the most important word in that phrase).

- **Standard Rule:** VP → V NP
- **Lexicalized Rule:** VP(*ate*) → V(*ate*) NP(*pizza*)

Why this fixes it: Now the probability depends on the words. The model can learn that

P(VP(*ate*) → V(*ate*) | NP(*pizza*)) is very high

but P(VP(*sleep*) → V(*sleep*) | NP(*pizza*)) is almost zero.

Questions?

Dependency Parsing
(more commonly used alternative)

Dependency Parsing
(more commonly used alternative)

Instead of locating substitutable subphrases, we want to directly represent the relationship between words

Constituency Parsing vs. Dependency Parsing

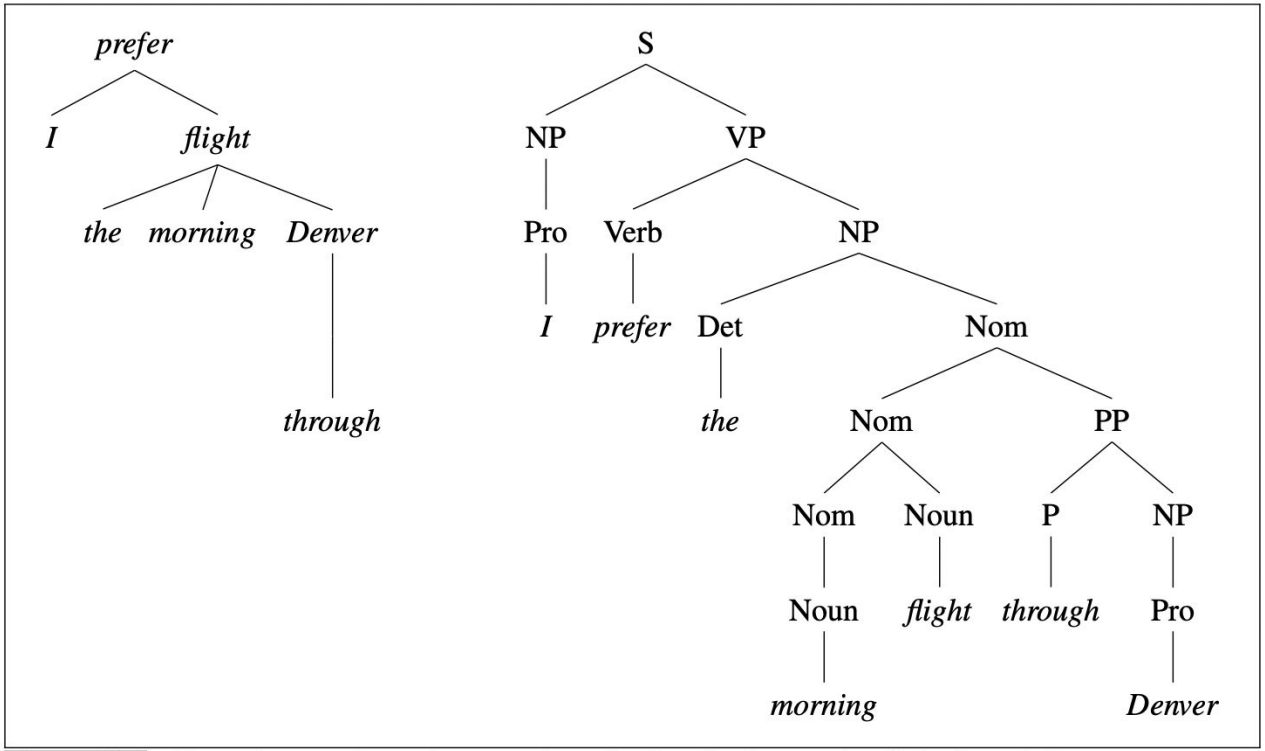


Figure 18.1 Dependency and constituent analyses for *I prefer the morning flight through Denver.*

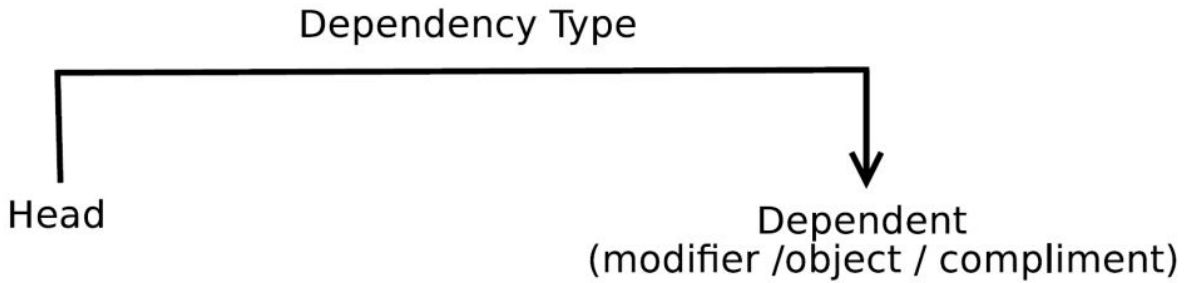
- What do these sentences have in common?
  - Kim gave the book to Sandy.
  - Kim gave Sandy the book.
  - The book was given to Sandy by Kim.
  - This is the book that Kim gave to Sandy.
  - Which book did Kim give to Sandy?
  - Kim will be expected to continue to try to give the book to Sandy.
  - This book everyone agrees Pat thinks Kim gave to Sandy.
  - This book is difficult for Kim to give to Sandy.

Constituent & Dependency Structures

- Kim gave the book to Sandy.
  - (S (NP Kim) (VP (V gave) (NP (D the) (N book)) (PP (P to) (NP Sandy))))
  - subj(gave, Kim)
  - dobj(gave, book)
  - iobj(gave, to)
  - dobj(to, Sandy)
  - spec(book, the)

Dependency Grammars

- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies

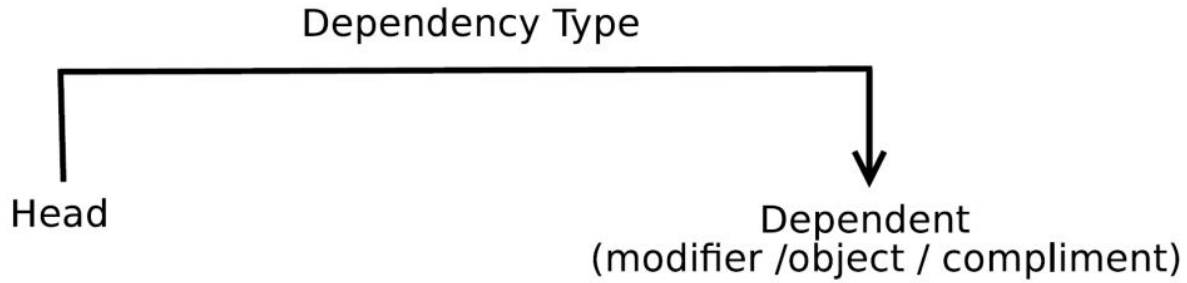


Dependency Grammars

- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies

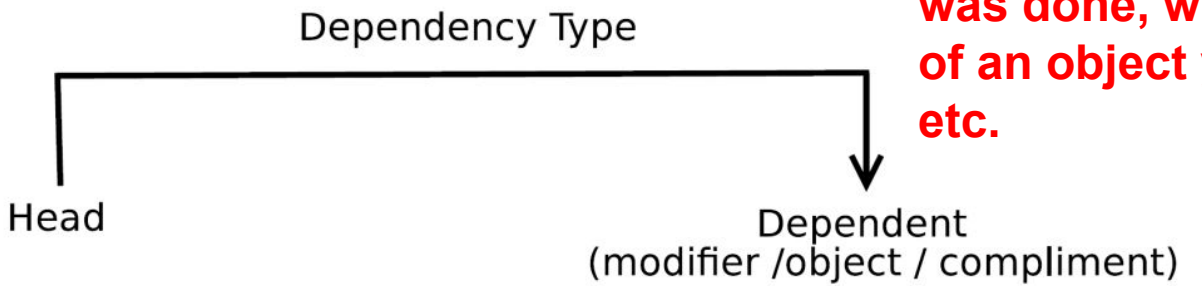
Binary: The relation involves exactly **two** words. One is the "Parent" (Head) and one is the "Child" (Dependent). **Asymmetrical:** The relationship only goes **one way**.

- If "Eating" is the head of "Apples," "Apples" cannot also be the head of "Eating."
- One word governs, and the other is governed.



Dependency Grammars

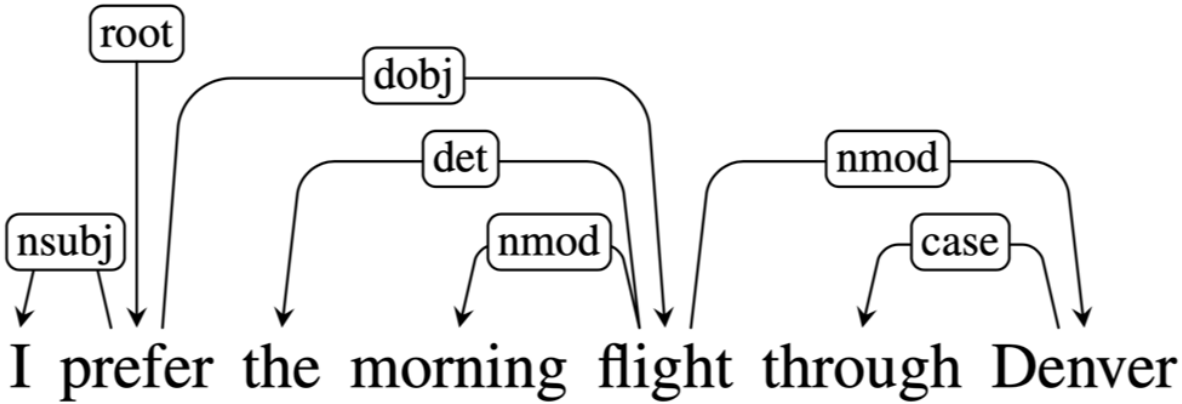
- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies



Dependent tells you something about the head: where an action was done, what kind of an object you have, etc.

Dependency Structures

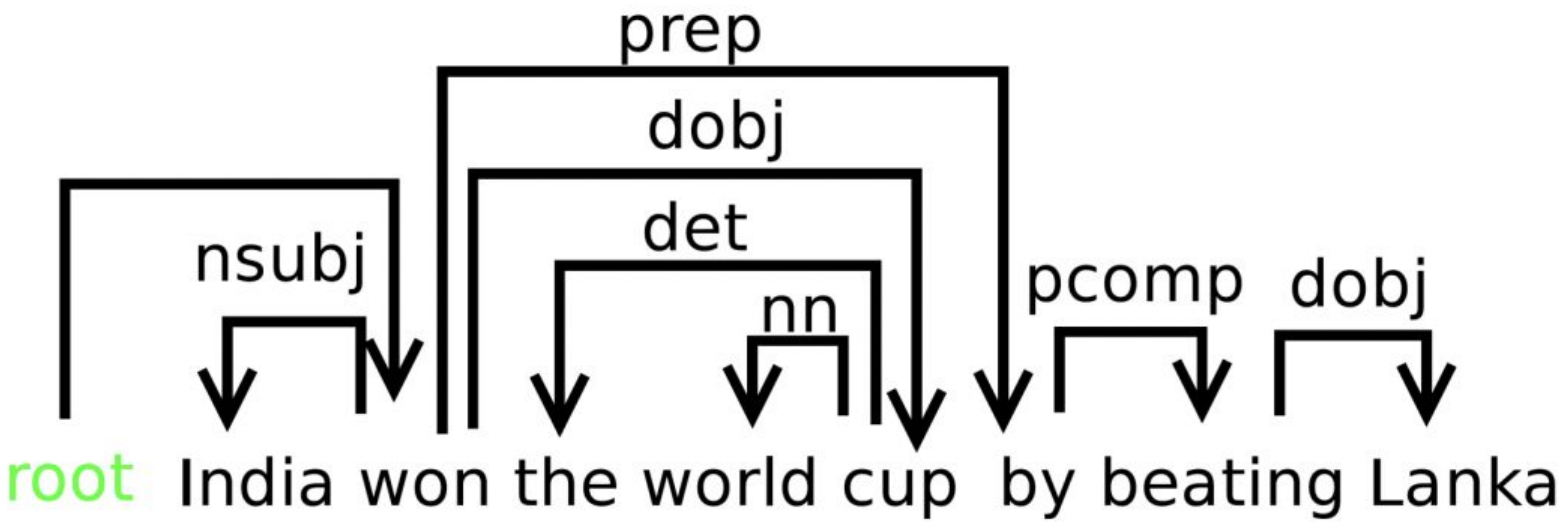
- Goal is to make the relationship between words explicit (vs. CFG)
- Typically represented as graphs, where each “vertex” is a word in the sentence
- Mandatory ROOT (attached to the main verb)
- Arcs labeled with the type of dependency



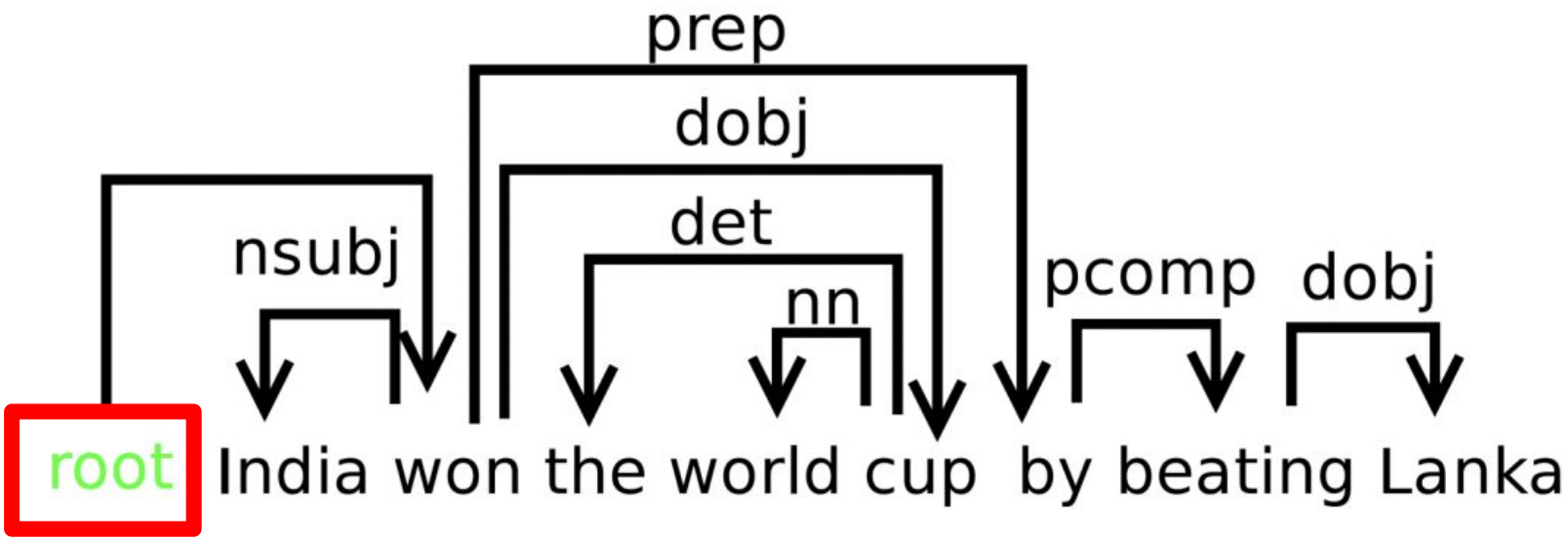
Dependency vs. CFG

Feature	CFG (CKY)	Dependency (Shift/Reduce)
Focus	**Groups** (Phonological chunks)	**Relations** (Grammatical roles)
Labels	Categories (NP, VP, PP)	Functions (Subject, Object, Mod)
Link	Word → Phrase → Sentence	Word → Word
Transparency	Implicit (Must derive roles)	**Explicit** (Directly labeled)

Example Dependency Parse

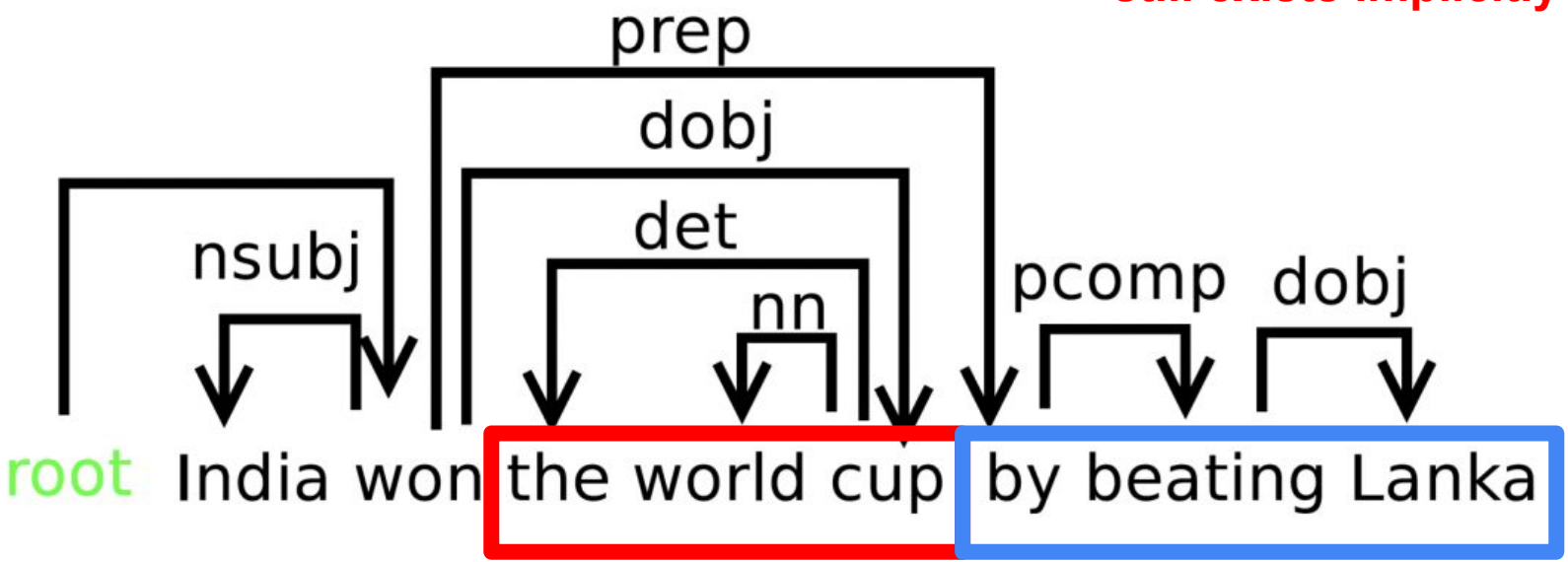


Example Dependency Parse



Example Dependency Parse

Notion of phrase still exists implicitly



Dependency Structures

Dependency structures are represented as graphs G = (V, A)

- (V)ertices and (A)rcs

V is roughly the set of words (+ punctuation)

- ❏ In a standard English sentence like "She ran.", the vertices are {She, ran, .}
- ❏ Every single word and every punctuation mark gets its own position in the graph and can have a relationship with another word.

Dependency Structures

In morphologically rich languages, includes stems and affixes too

- In languages like Turkish, Finnish, or Hungarian, a single "word" can contain the meaning of an entire English sentence because of prefixes and suffixes (affixes).
- Instead of making one node for the giant word, the parser breaks the word into its **Stem** (the core meaning) and its **Affixes** (the grammar bits).
- If a word means "in my houses," the parser might create separate nodes for "house," "plural," and "in." This allows the grammar to show the relationship between the "plural" part and the "house" part explicitly.

Where do arcs come from?

The names of the arcs (like `nsubj`, `dobj`, `nn`, `pcomp`) come from a **Dependency Taxonomy**.

- Most modern parsers use **Universal Dependencies (UD)**, which is a standardized list of about 40 relations that work for all human languages.
- Linguists defined these based on traditional grammar.

Universal Dependencies Project

- https://universaldependencies.org/
- Treebank = corpora with sentences that have dependency parses
- Set of dependency labels that are:
  - Linguistically motivated
  - Computationally useful
  - Cross-linguistically applicable
- 100+ dependency treebanks for over 60 languages

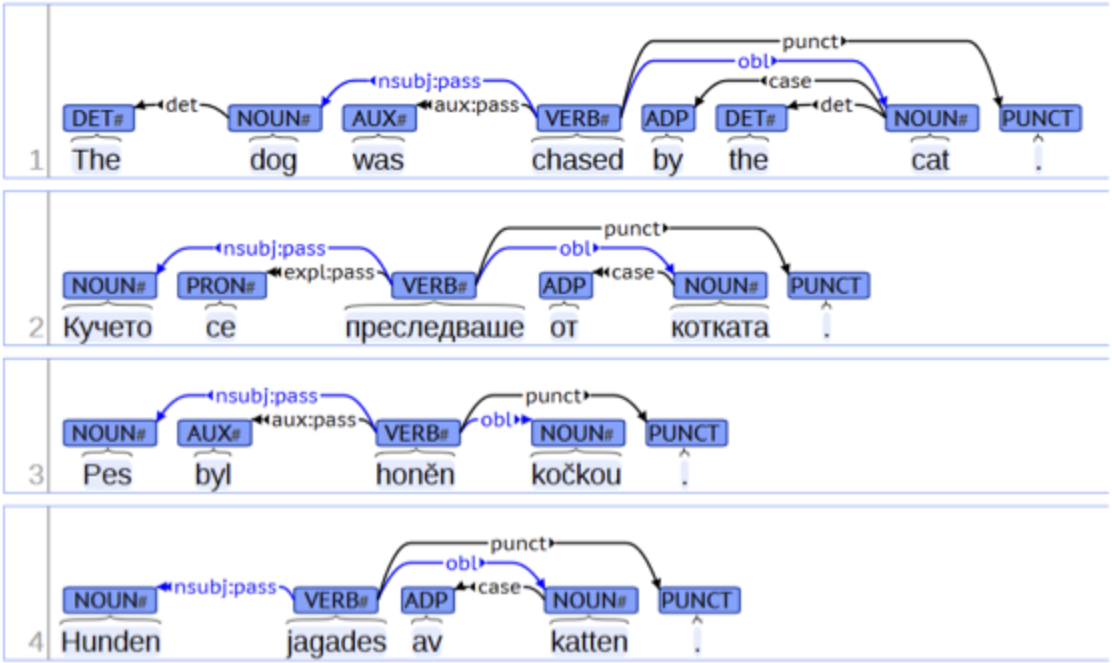
Universal Dependency relations

Clausal Argument Relations	Description
NSUBJ	Nominal subject
OBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 18.2 Some of the Universal Dependency relations (de Marneffe et al., 2021).

Universal Dependencies Illustrated

Parallel examples for English, Bulgarian, Czech & Swedish



https://universaldependencies.org/introduction.html

Dependency Parsing

- **Goal:** Learn a good predictor of dependency graphs
- **Input:** Sentence
- **Output:** Dependency graph/tree G = (V, A)
- We will study transition-based dependency parsing

Transition-based dependency parsing

Architecture

- **The Stack:** A "work-in-progress" area. We only look at the top two items.
- **The Buffer:** The "conveyor belt" holding the words of the sentence that haven't been processed yet.
- **The Arcs:** A list where we record the relationships (arrows) we find.
- **The Classifier:** The "brain" that looks at the current Stack and Buffer and chooses the next move.

The parser’s action menu

SHIFT: Move the next word from the **Buffer** to the **Stack**.

LEFT-ARC (L-ARC): Create an arrow from the top word (S1) to the second word (S2).

- **S1 → S2**
- **Pop S2** (the dependent) from the stack.

RIGHT-ARC (R-ARC): Create an arrow from the second word (S2) to the top word (S1).

- **S2 → S1**
- **Pop S1** (the dependent) from the stack.

Dependency Parsing

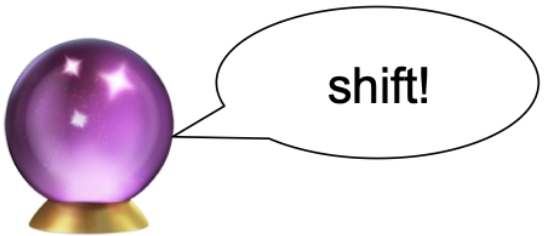
Book me the morning flight

```
stack = [root]

buffer = [book, me, the, morning, flight]

relations = []
```

Dependency Parsing

Book me the morning flight

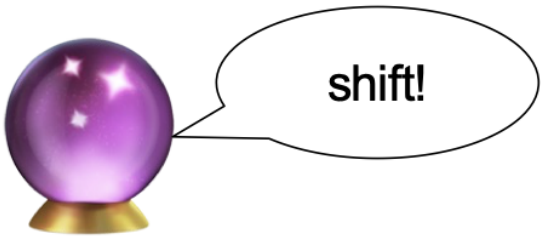
```
stack = [root]

buffer = [book, me, the, morning, flight]

relations = []
```



Dependency Parsing

Book me the morning flight

```
stack = [root]

buffer = [book, me, the, morning, flight]

relations = []
```

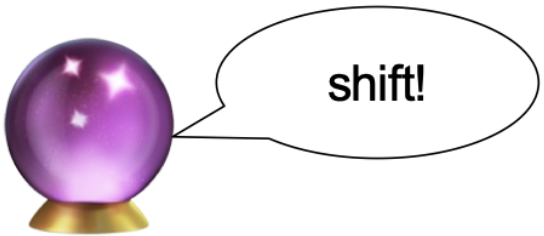


Step 0: when there are less than 2 things on the stack, always shift.

Dependency Parsing

Book me the morning flight

```
stack = [root]
buffer = [book, me, the, morning, flight]
relations = []
```

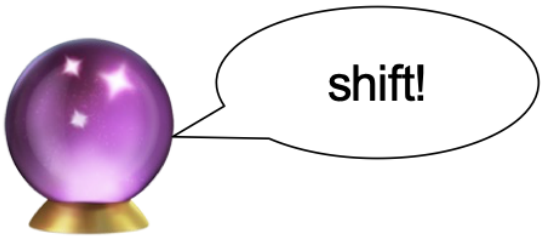


Step 0: when there are less than 2 things on the stack, always shift.

Dependency Parsing

Book me the morning flight

```
stack = [root, book]
buffer = [me, the, morning, flight]
relations = []
```

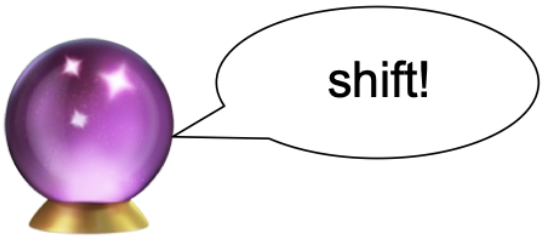


Step 0: when there are less than 2 things on the stack, always shift.

Dependency Parsing

Book me the morning flight

```
stack = [root, book, me]
buffer = [the, morning, flight]
relations = []
```

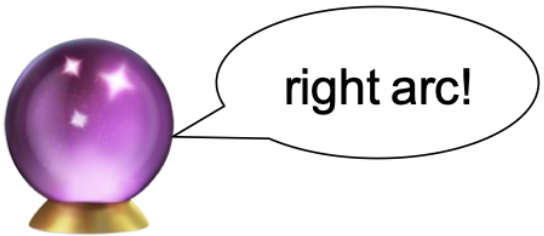


Step 1

Dependency Parsing

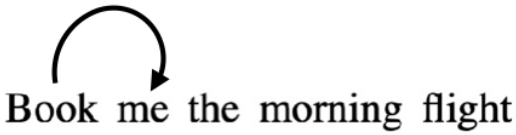
Book me the morning flight

```
stack = [root, book, me]

buffer = [the, morning, flight]

relations = []
```

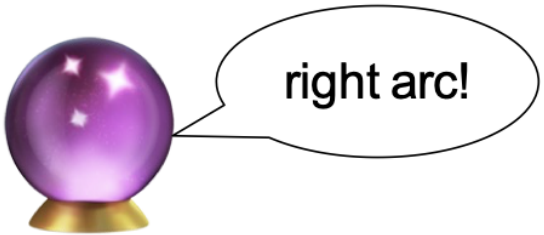


Step 2: assign book as head of me and remove me from stack

Dependency Parsing

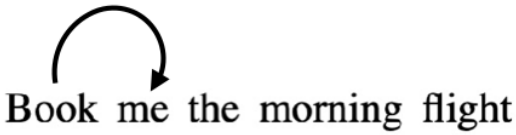



```
stack = [root, book]
buffer = [the, morning, flight]
relations = [book->me]
```

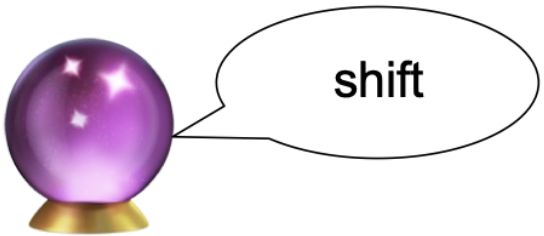


Step 2: assign book as head of me and remove me from stack

Dependency Parsing

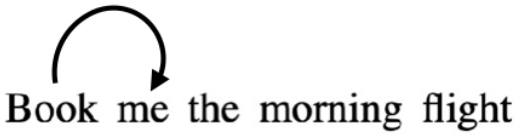



```
stack = [root, book, the]

buffer = [morning, flight]

relations = [book->me]
```

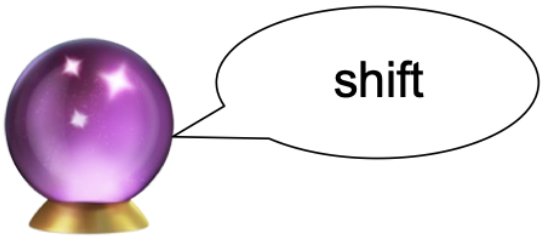


Step 3

Dependency Parsing

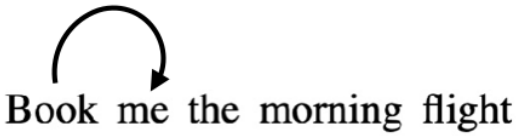



```
stack = [root, book, the, morning]

buffer = [flight]

relations = [book->me]
```

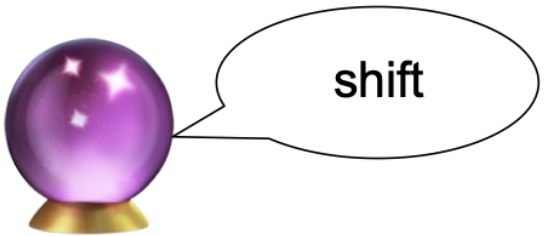


Step 4

Dependency Parsing

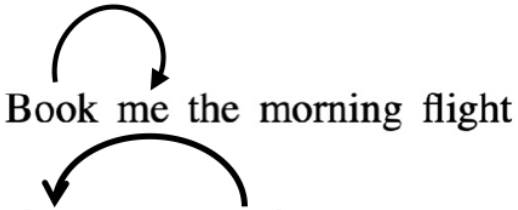



```
stack = [root, book, the, morning, flight]

buffer = []

relations = [book->me]
```

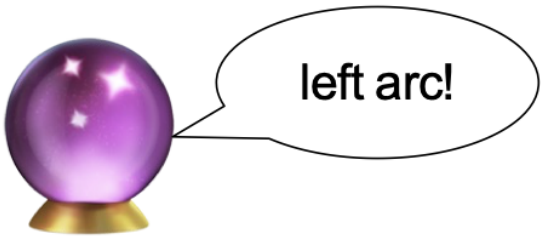


Step 5

Dependency Parsing

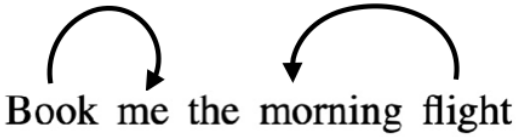



```
stack = [root, book, the, morning, flight]

buffer = []

relations = [book->me]
```

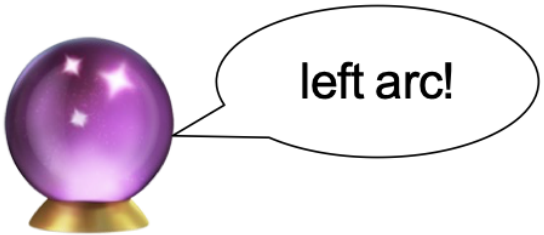


Step 6: assign flight as head of morning, remove morning

Dependency Parsing

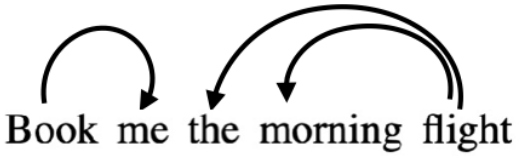



```
stack = [root, book, the, flight]

buffer = []

relations = [book->me, flight->morning]
```

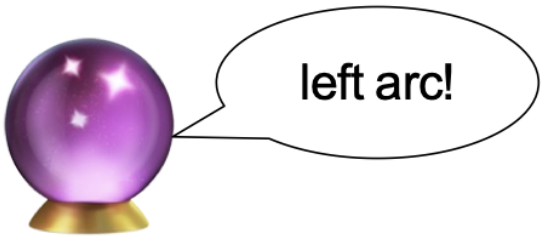


Step 6: assign flight as head of morning, remove morning

Dependency Parsing

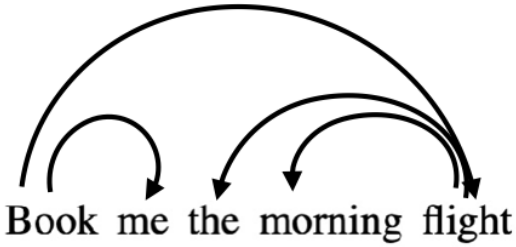



```
stack = [root, book, flight]

buffer = []
relations = [book->me, flight->morning,
             flight->the]
```

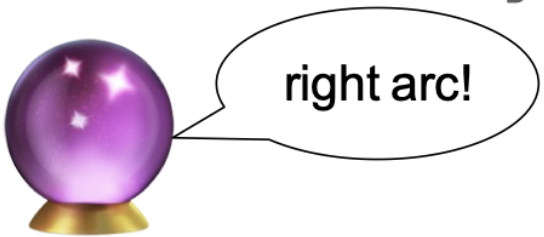


Step 7: assign flight as head of the, remove the

Dependency Parsing

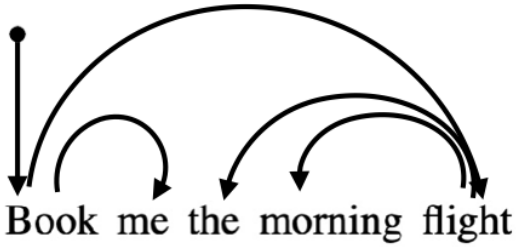



```
stack = [root, book]

buffer = []
relations = [book->me, flight->morning,
             flight->the, book->flight]
```

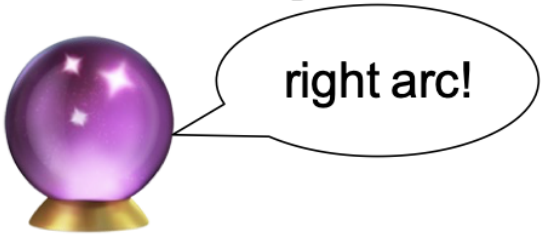


Step 8: assign book as head of flight, remove flight

Dependency Parsing

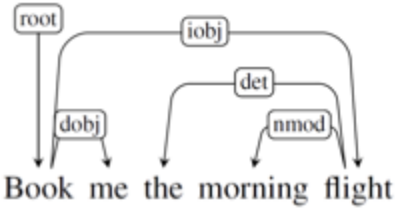



```
stack = [root]

buffer = []
relations = [book->me, flight->morning,
     flight->the, book->flight, root->book]
```



Step 9: assign root as head of book, remove book

Transition-Based Parsing Illustrated



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 14.7 Trace of a transition-based parse.

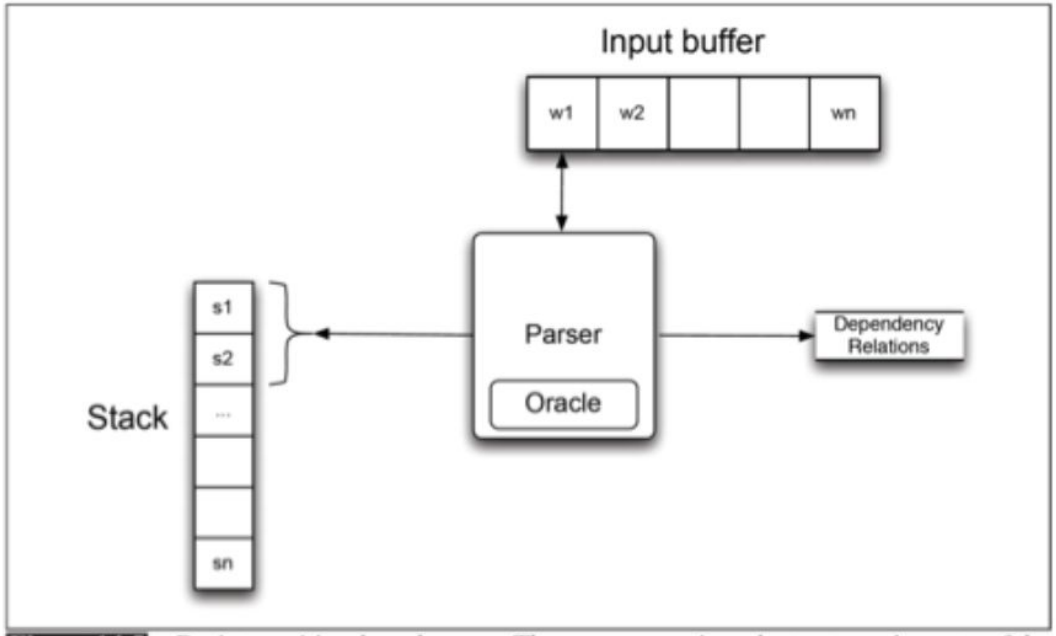


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

The Oracle: Defining the "Gold" Standard

The component that tells the parser the "correct" move (Shift, Left-Arc, Right-Arc) for any given state.

- **During Training:** The Oracle is the "Gold Standard." We take a perfectly parsed sentence (from a treebank) and work backward to see which sequence of moves would have created it. This sequence is the "Oracle path."
- **During Inference (Testing):** The Oracle tries to predict the "Gold" move based on features it learned during training.

Oracle: Supervised Machine Learning

Classification problem:

- **Input:** Current and previous configurations
- **Output:** Which transition to take

The usual ML options are available:

- Feature-based classifiers (Logistic regression, Naive Bayes, etc.)
- Word-embedding features
- Neural network classifiers…

But, we still need training data & features to learn from

Training data for oracle

The Source: Treebanks (like Universal Dependencies). These are collections of thousands of sentences manually parsed by linguists.
The Conversion: We can't feed a tree directly into a classifier. We must use an **Oracle Function** to work backward.

- We take a Gold Tree and simulate the parsing process.
- At each step, we ask: "What move (Shift, L-Arc, R-Arc) would lead toward this gold tree?"

The Result: A training set consisting of **(Configuration, Action)** pairs.

- **Configuration:** The current state of the Stack and Buffer.
- **Action:** The "correct" move to take.

Feature representation

The classifier doesn't "read" the whole sentence at once. it looks at a "window" of information.

Top Features:

- **Word Forms:** The actual words at the top of the stack
- **POS Tags:** The Part-of-Speech tags for those same words

Complex Features:

- **Morphology:** Gender, Number, or Case (crucial for "rich" languages).
- **Tree Structure:** The labels of the children already attached to the words (e.g., "Does the word on the stack already have a subject?").

Vectorization: These features are turned into a long string of numbers (a vector) and fed into a Neural Network or SVM

The Greedy Trap: Error Propagation

- **No Backtracking:** Transition-based parsing is a "one-shot" process. Once a move is made, it cannot be undone.
- **The "Domino Effect":** Because the parser is greedy, a single mistake by the Oracle at the start of a sentence "corrupts" the Stack.
- **Compounding Errors:**
  - Step 2: Oracle makes 1 wrong move.
  - Step 3: The parser is now in a state that never existed in its training data.
  - Step 4+: Every subsequent move is likely to be wrong.

Total tree quality is highly sensitive to the Oracle's accuracy on early decisions.

Modern NLP: Parsing in the Era of LLMs

No Explicit Step: Unlike CKY or Shift-Reduce, LLMs (like GPT-4) do not have a dedicated "Parsing" phase.

Implicit Syntax: As the model processes text, it builds an internal mathematical representation of structure through **Self-Attention**.

Attention = Connection: The model calculates "weights" between words.

- *Example:* In "The **chef** who won the award **cooked**...", the Attention mechanism "links" the subject to the verb across the long distance.

Parsing in the era of LLMs

Hidden Structural Layers

Research (often called "BERTology") has shown that as data passes through the many layers of a Transformer, the internal layers actually build a "virtual" parse tree.

- **Lower Layers:** Tend to focus on local syntax (like POS tagging).
- **Middle Layers:** Tend to capture long-range dependencies (like Subject-Verb agreement).
- **Higher Layers:** Focus on complex semantics and meaning.

Generation vs. Grammar

While the CFGs we studied are **Prescriptive** (they tell you what is "grammatical" or "ungrammatical"), LLMs are **Probabilistic**.

- An LLM doesn't reject a sentence for being ungrammatical; it just assigns it a lower probability.
- It "learns" grammar by predicting the next token millions of times until it implicitly understands that an "NP" is usually followed by a "VP."

Summary

Feature	Traditional Parsing (Slides)	LLM "Parsing"
Rules	Explicit (CFG, Lexicon)	Implicit (Learned from Data)
Structure	Explicit Trees (Constituency/Dependency)	Hidden Vector States
Failure	Crashes if "Ungrammatical"	Always produces a result (even if messy)
Mechanism	Algorithms like CKY or Shift-Reduce	Multi-Head Self-Attention

Application: Information Extraction

- Parsing allows us to extract structured knowledge (Triples).
- We extract: (Subject, Relation, Object).
- Example: 'Steve Jobs founded Apple.'
- Dependency Parser identifies: nsubj(Jobs, founded) and obj(Apple, founded).
- Result: (Steve Jobs, founded, Apple).

Application: Sentiment Analysis

• Parsing resolves which entity a sentiment refers to (Fine-grained sentiment).
• Example: 'The screen is great but the battery is bad.'
• A simple model might just see 'great' and 'bad' and get confused.
• Dependency trees explicitly link the adjective 'great' to 'screen' and 'bad' to 'battery'.

Application: Semantic Parsing (SQL)

• Transforming a natural language question into a database query.
• Example: 'Who directed Inception?'
• The parser identifies 'directed' as the relation and 'Inception' as the object.
• Maps to: SELECT director FROM movies WHERE title='Inception'.

Application: Machine Translation

- Syntactic Transfer: Different languages have different grammatical structures.
- English is S-V-O (Subject-Verb-Object). Japanese is S-O-V.
- Parsing the source sentence helps the system structurally re-order the constituents correctly before translating the words.