Constituency Grammars and Constituency Parsing

Natalie Parde

UIC CS 421

POS tags are one way to formalize language structure.

- Constituency grammars are another!
- Constituency grammars are:
  - A **set of rules** that describe how a language can be structured
  - A **lexicon** that defines the words and symbols that belong to the language

Constituency Grammars

- Break sentences into hierarchical parts
- Provide the necessary structure to answer important questions:
  - What are the **constituents** (groups of words that behave as a single unit or phrase) in this sentence?
  - What are the **grammatical relations** between these constituents?
  - Which words are **dependent** upon one another?
- Although most approaches we've seen model sentences as sequences, **formal grammars model sentences as recursive generating processes**
  - Usually, this is done using a tree structure

It's all about finding the right balance!

Overgeneration:	English:	Undergeneration:
Love NLP class my so much that don't care about being it early morning in!	I love my NLP class so much that I don't even care about it being in early morning!	I love my class!
Did get the you email guy that that from class said he forward to you would?	Did you get the email that that guy from class said he would forward to you?	Did you get his email?
Well, there just happened.	Well, that just happened.	What happened?

This Week's Topics

Tuesday

- Context-Free Grammars
- Syntactic Parsing
- CKY Algorithm

Thursday

- Earley Algorithm
- Probabilistic CKY
- Lexicalized Grammars

This Week's Topics

Tuesday

- Context-Free Grammars
- Syntactic Parsing
- CKY Algorithm

Thursday

- Earley Algorithm
- Probabilistic CKY
- Lexicalized Grammars

Grammar Formalisms vs. Specific Grammars

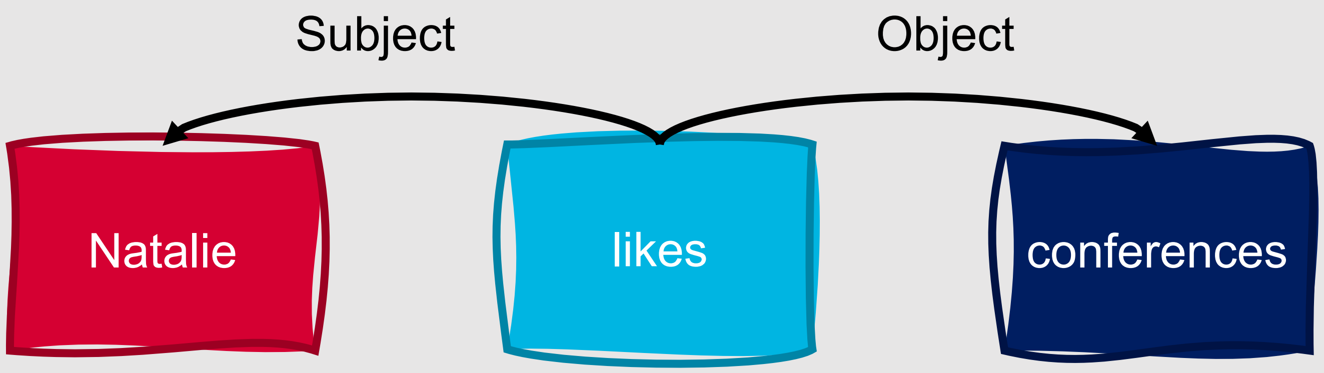
- **Grammar Formalisms:** A precise way to define and describe the structure of independent sentences.
  - There are many different grammar formalisms (you can learn much more about these in linguistics courses!)
- **Specific Grammars:** Implementations (according a specific formalism) for a particular language
  - English, Arabic, Mandarin, or Hindi
- Grammar Formalisms : Specific Grammars :: Programming Languages : Programs
- In general, our specific grammars are close but imperfect ways to formalize a language
  - For example: There are an infinite number of possible English sentences, but our specific grammar for English needs to be finite

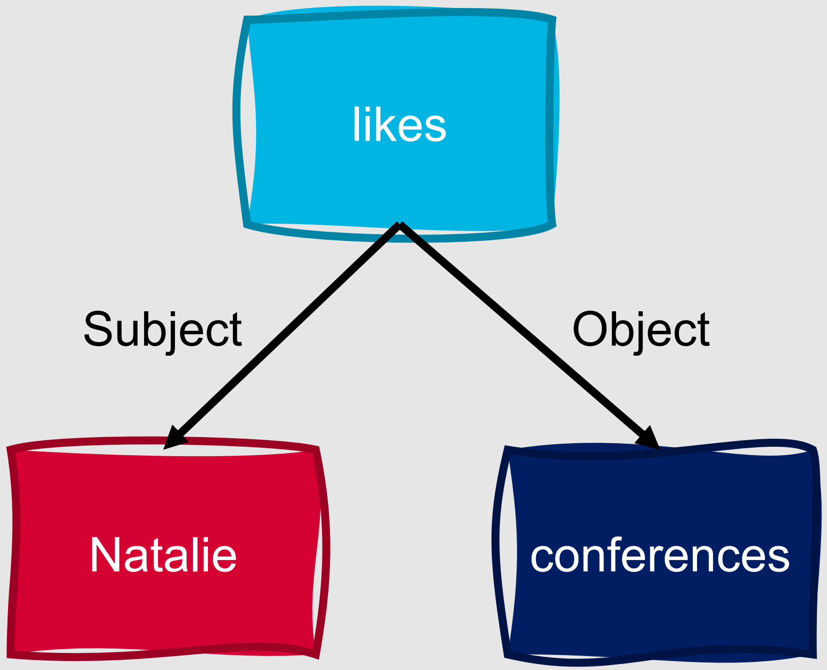
Basic English Sentence Structure

Natalie	likes	conferences
Noun (Subject)	Verb (Head)	Noun (Object)

There are many ways to represent a sentence!

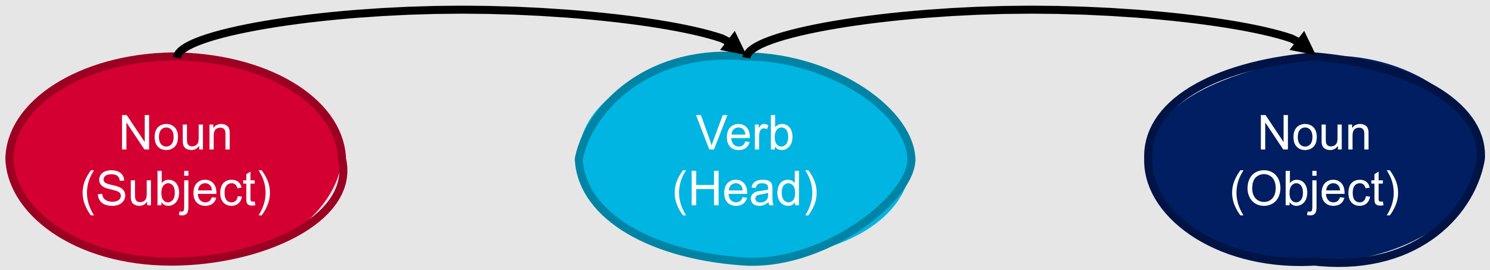
As a dependency graph:





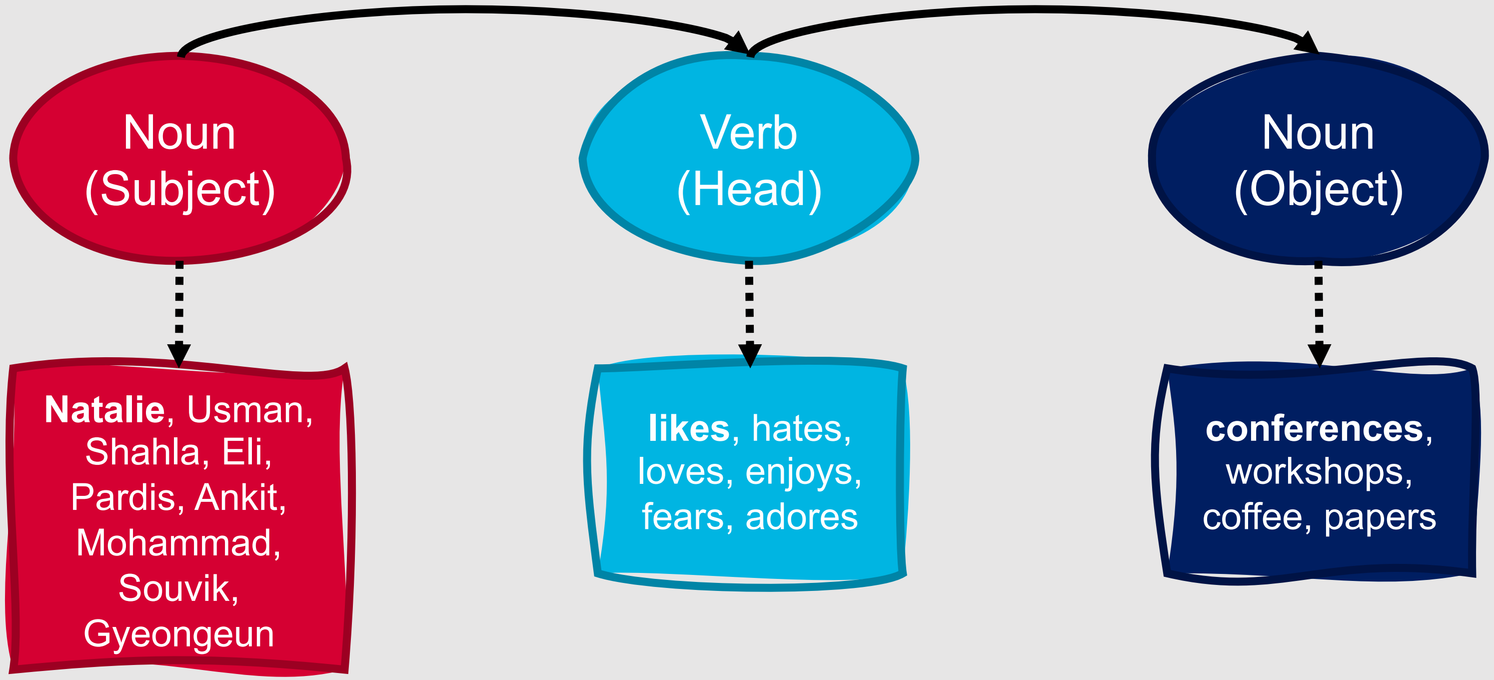
There are many ways to represent a sentence!

As a finite state automaton:



There are many ways to represent a sentence!

As a hidden Markov model:



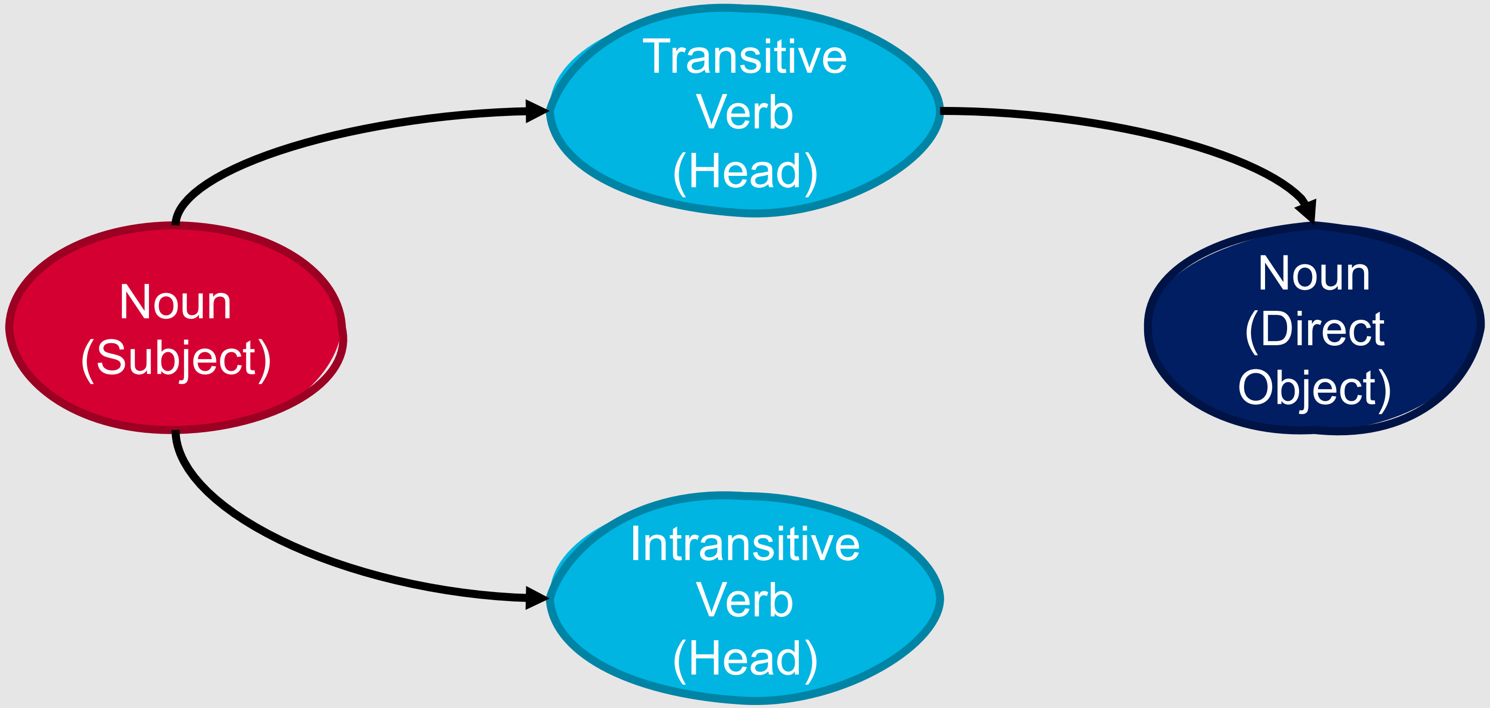
Different types of words accept different types of arguments.

- **Subcategorization:** Syntactic constraints on the set of arguments that a group of words will accept.
  - **Intransitive verbs** accept only subjects
    - Sleep, arrive
  - **Transitive verbs** accept a subject and a direct object
    - Eat, drink
  - **Ditransitive verbs** accept a subject, a direct object, and an indirect object
    - Give, make
- **Selectional Preference:** Semantic constraints on the set of arguments that a group of words will accept.

Natalie likes conferences. 🙂

Natalie drinks conferences. 🤨

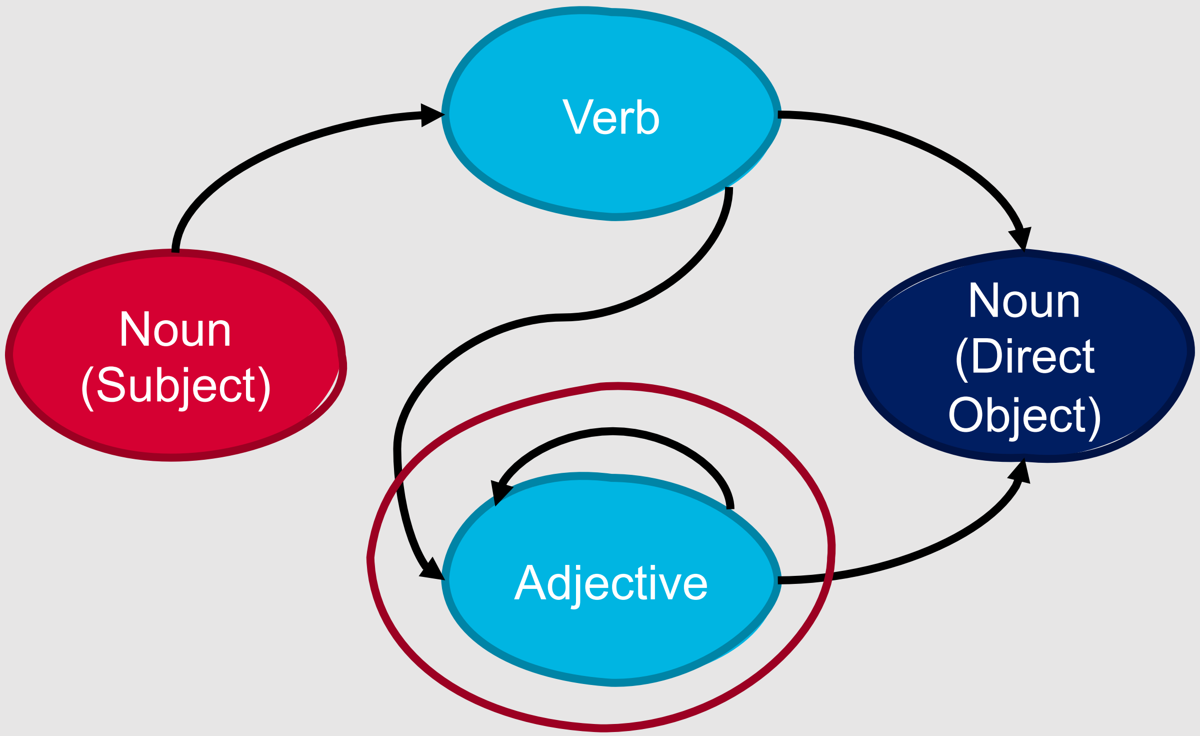
We might represent these as a finite state model like this:



One of the reasons why the number of possible English sentences is infinite?

- Language is recursive!
- In theory, we can have unlimited modifiers (adjectives and adverbs)
  - Natalie likes conferences.
  - Natalie likes academic conferences.
  - Natalie likes busy academic conferences.

We can easily model simple cases of recursion in a finite state model as well.



However, recursion in sentences can also be more complex.

Natalie likes conferences.

Natalie likes conferences **in Europe**.

Natalie likes conferences **in Europe in the summer**.

Still, can't we just make complex FSAs?

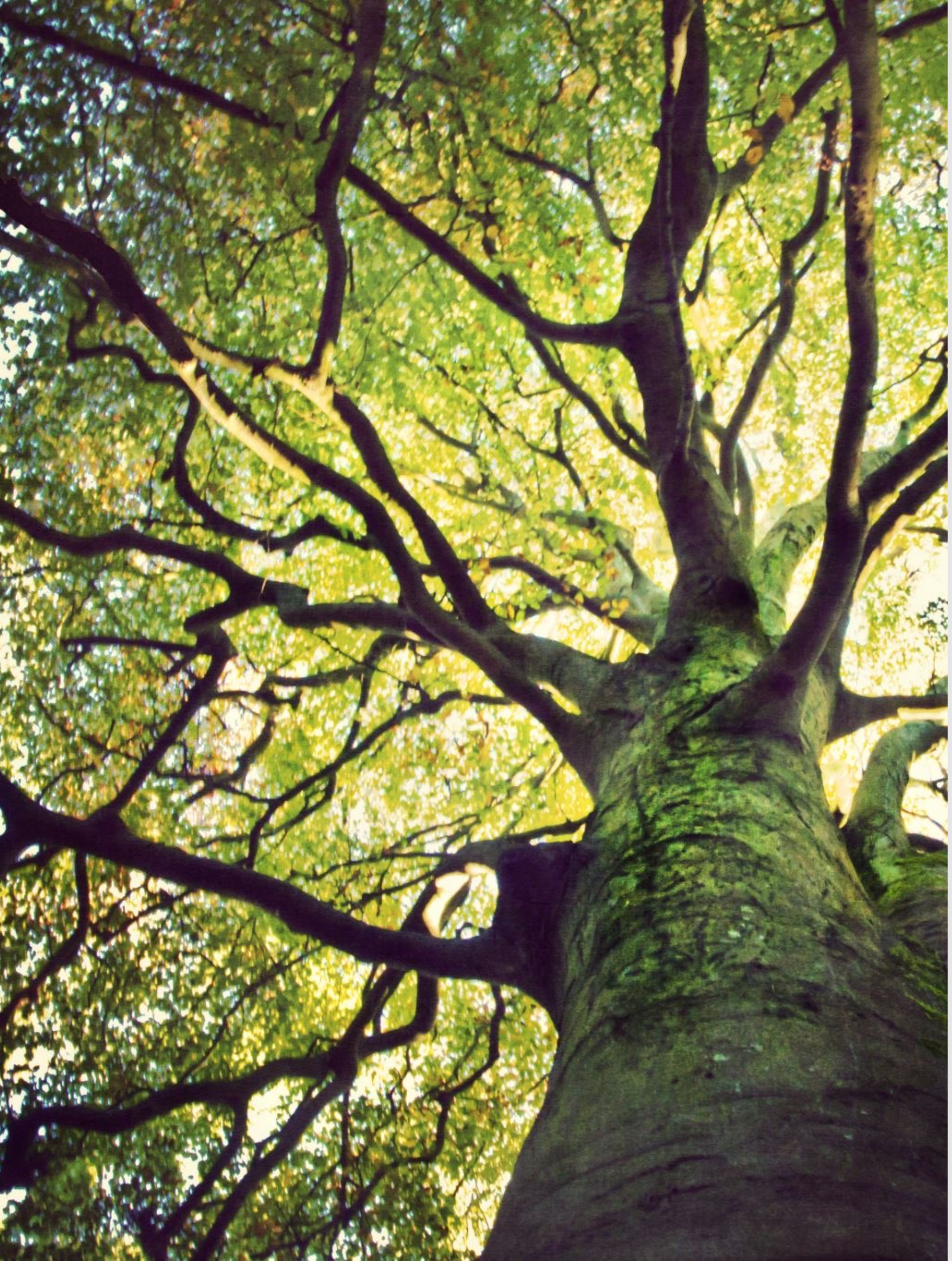
- FSAs can model recursion, but they can't model hierarchical structure or handle issues like **attachment ambiguity**

Natalie likes conferences in either Europe or Asia.

Natalie **likes conferences in** Europe **or** Asia.

Natalie **likes** conferences in Europe **or** Asia.

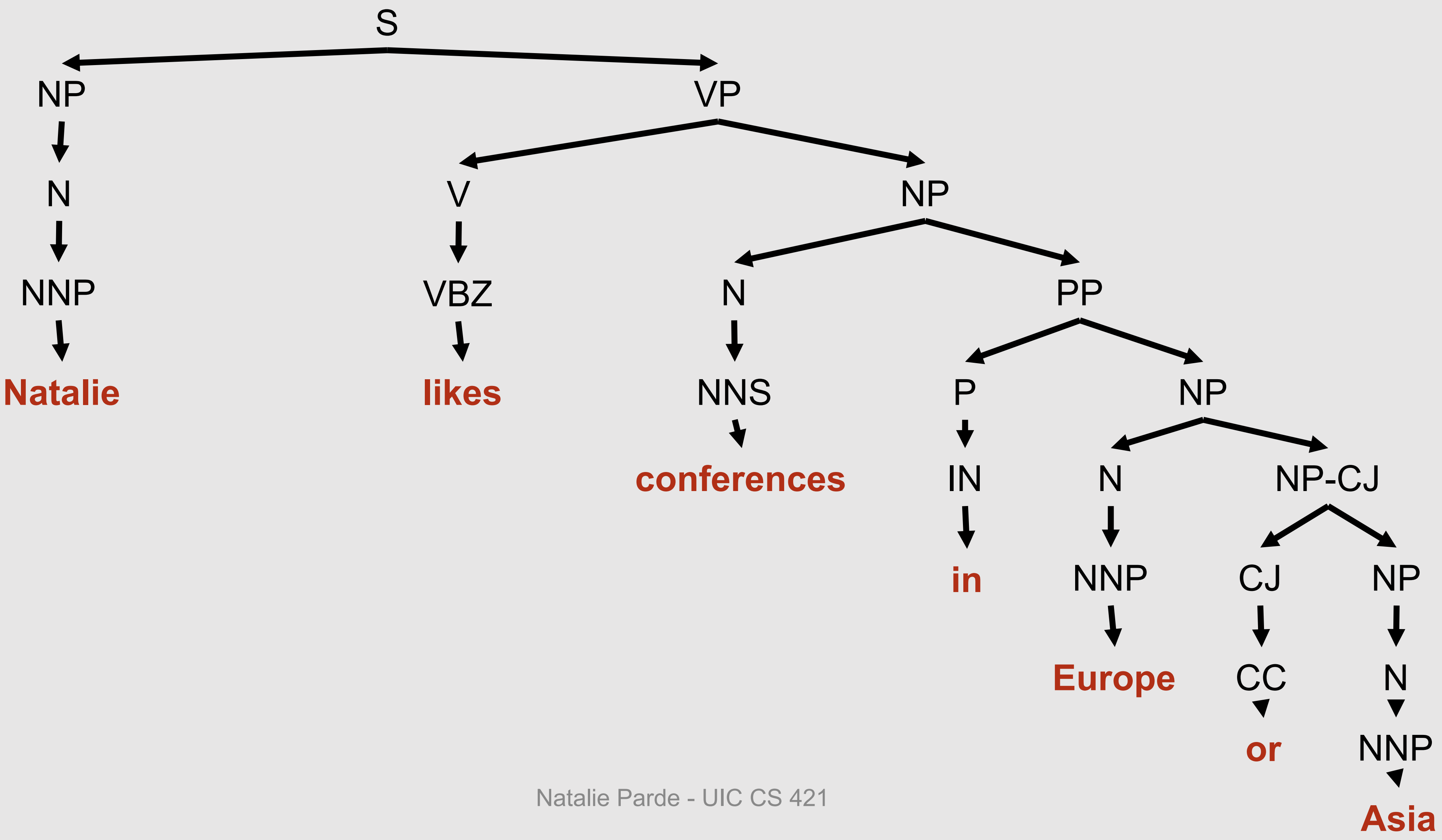
Natalie likes two things: Asia, or conferences in Europe.



Hierarchical trees to the rescue!

- A sentence consists of words that can be grouped into phrases (**constituents**) using a hierarchical structure
- Formal trees will usually have **internal (non-terminal) nodes** and **outer (terminal) leaves**
- **Nodes: Elements of sentence structure**
  - Constituent type
  - POS type
- **Leaves: Surface wordforms**
- The nodes and leaves are connected to one another by **branches**

What does this look like?



The grammars defining these hierarchical trees are context-free grammars.

- **Context-Free Grammar (CFG):** A mathematical system for modeling constituent structure in regular languages.
- CFGs are defined by productions that indicate which strings they can generate.
  - **Production:** Rules expressing the allowable combinations of symbols (e.g., POS types) that can form a constituent
  - Productions can be **hierarchically embedded**
    - Noun Phrase (NP) → Determiner Nominal
    - Nominal → Noun | Nominal Noun
- Why is it called context-free?
  - A subtree can be replaced by a production rule independent of the greater context (other nodes in the hierarchy) in which it occurs.
- Also called **Phrase-Structure Grammars**

Formal Definition

- A CFG is a 4-tuple $\langle \boldsymbol{N}, \boldsymbol{\Sigma}, \boldsymbol{R}, \boldsymbol{S} \rangle$ consisting of:
  - A set of non-terminal nodes ***N***
    - ***N*** = {S, NP, VP, PP, N, V, …}
  - A set of terminal nodes (leaves) $\boldsymbol{\Sigma}$
    - $\boldsymbol{\Sigma}$ = {time, flies, like, an, arrow, …}
  - A set of rules ***R***
  - A start symbol ***S*** ∈ ***N***
- How to check for **grammatical correctness**?
  - Any sentences for which the CFG can construct a tree (all words in the sentence must be reachable as leaf nodes) are accepted by the CFG.

Production rules determine how constituents can be combined.

Constituent: A group of words that behaves as a single unit.

- Noun Phrase: the woman, the woman with red hair, the last conference of the year
- Prepositional Phrase: with red hair, of the year
- Verb Phrase: drinks tea, likes going to conferences

Constituents contain **heads** and **dependents**

- **Heads:** the *woman* with red hair, the last *conference* of the year
- **Dependents:** *the* woman *with red hair*, *the last* conference *of the year*

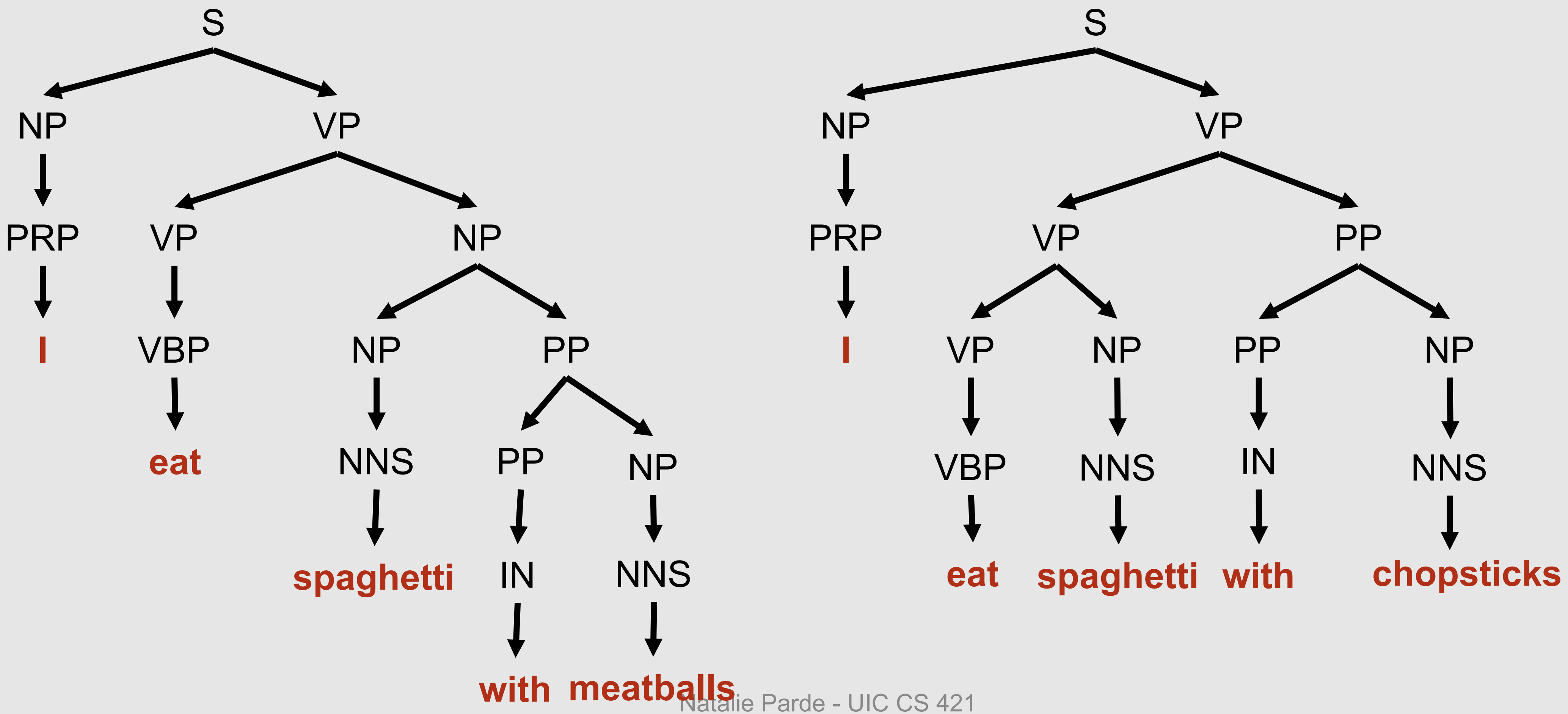
Dependents can be arguments or adjuncts

- Arguments are **obligatory**
  - Natalie likes *conferences*. 🙂
  - Natalie likes. 🤨
- Adjuncts are **optional**
  - Natalie drinks *tea*. 🙂
  - Natalie drinks. 🙂

Properties of Constituents

- **Constituents can be substituted with one another** in the context of the greater sentence
  - **The woman with red hair** rolled her eyes as lightning immediately struck the man's house.
  - **The unicorn** rolled her eyes as lightning immediately struck the man's house.
- **A constituent can move around** within the context of the sentence
  - **The woman with red hair** rolled her eyes as lightning immediately struck the man's house.
  - Lightning immediately struck the man's house as **the woman with red hair** rolled her eyes.
- **A constituent can be used to answer a question** about the sentence
  - Who rolled her eyes? **The woman with red hair**.

The structure of constituents in a tree corresponds to their meaning.



Case Example

- Draw a constituent tree for the sentence:
  - **Time flies like an arrow.**

Production Rules	
S ! NP VP	PP ! P NP
NP ! DET N	PP ! P
NP ! N	P ! like
NP ! N N	V ! flies | like
VP ! VP PP	DET ! a | an
VP ! V NP	N ! time | fruit | flies | arrow | banana
VP ! V	

Case Example

<table>
<tr><th colspan="2">Production Rules</th></tr>
<tr><td>S ! NP VP</td><td>PP ! P NP</td></tr>
<tr><td>NP ! DET N</td><td>PP ! P</td></tr>
<tr><td>NP ! N</td><td>P ! like</td></tr>
<tr><td>NP ! N N</td><td>V ! flies | like</td></tr>
<tr><td>VP ! VP PP</td><td>DET ! a | an</td></tr>
<tr><td>VP ! V NP</td><td rowspan="2">N ! time | fruit | flies | arrow | banana</td></tr>
<tr><td>VP ! V</td></tr>
</table>

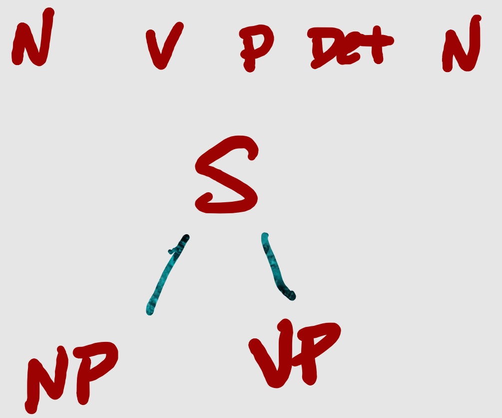
Time flies like an arrow

N V P Det N

Case Example

Production Rules	
S ! NP VP	PP ! P NP
NP ! DET N	PP ! P
NP ! N	P ! like
NP ! N N	V ! flies | like
VP ! VP PP	DET ! a | an
VP ! V NP	N ! time | fruit | flies | arrow | banana
VP ! V	

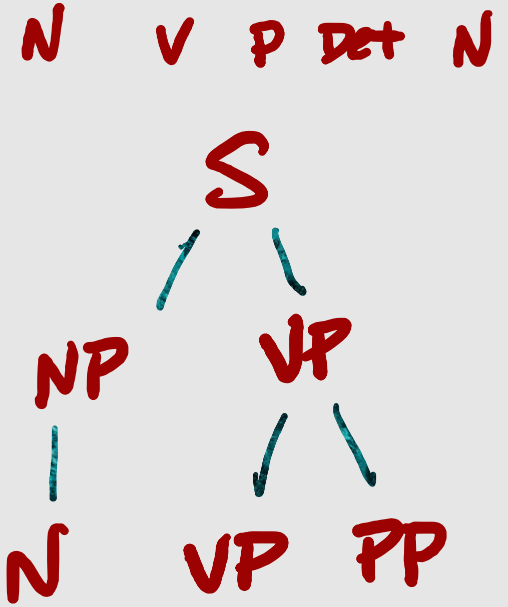
Time flies like an arrow



Case Example

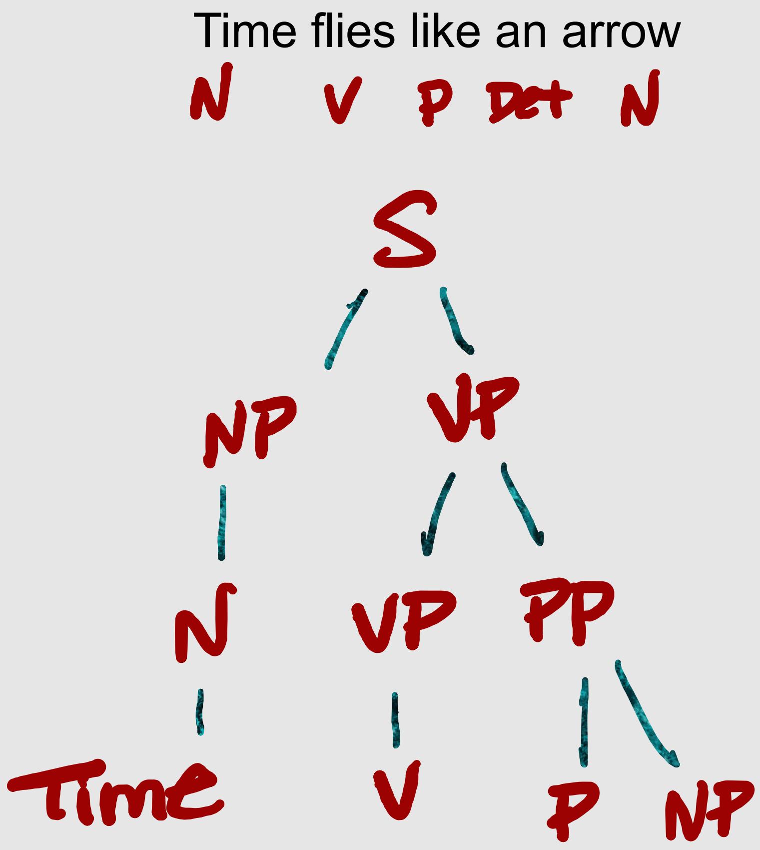
Production Rules	
S ! NP VP	PP ! P NP
NP ! DET N	PP ! P
NP ! N	P ! like
NP ! N N	V ! flies | like
VP ! VP PP	DET ! a | an
VP ! V NP	N ! time | fruit | flies | arrow | banana
VP ! V	

Time flies like an arrow



Case Example

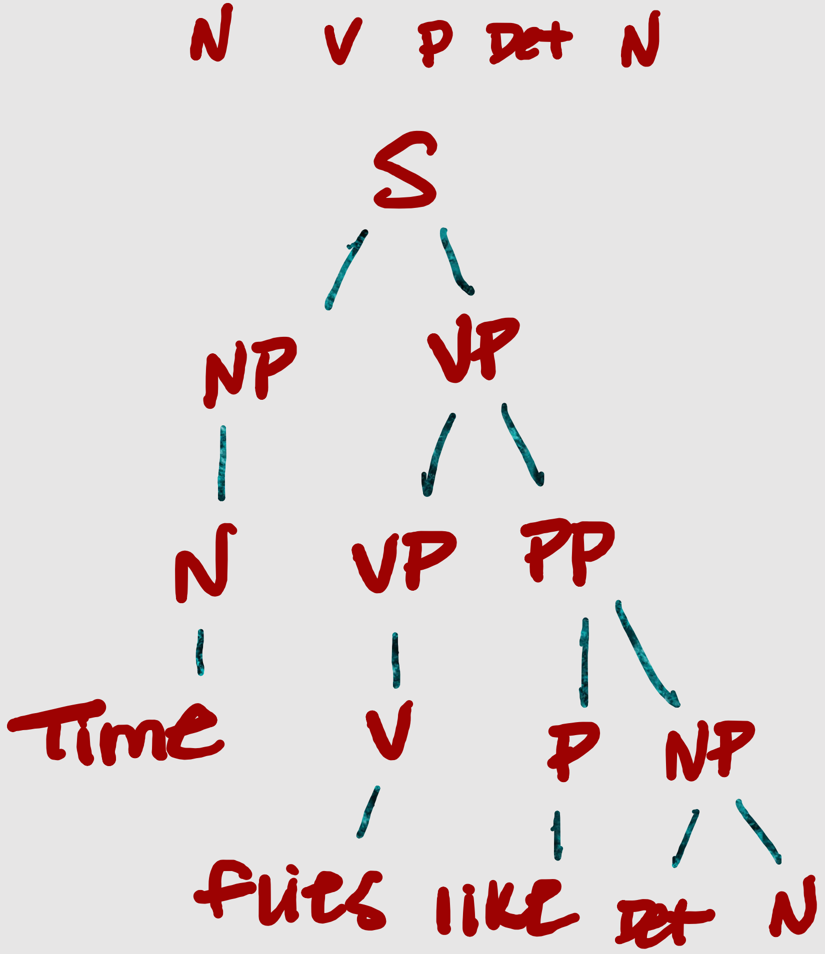
Production Rules	
S ! NP VP	PP ! P NP
NP ! DET N	PP ! P
NP ! N	P ! like
NP ! N N	V ! flies | like
VP ! VP PP	DET ! a | an
VP ! V NP	N ! time | fruit | flies | arrow | banana
VP ! V	



Case Example

Production Rules	
S ! NP VP	PP ! P NP
NP ! DET N	PP ! P
NP ! N	P ! like
NP ! N N	V ! flies | like
VP ! VP PP	DET ! a | an
VP ! V NP	N ! time | fruit | flies | arrow | banana
VP ! V	

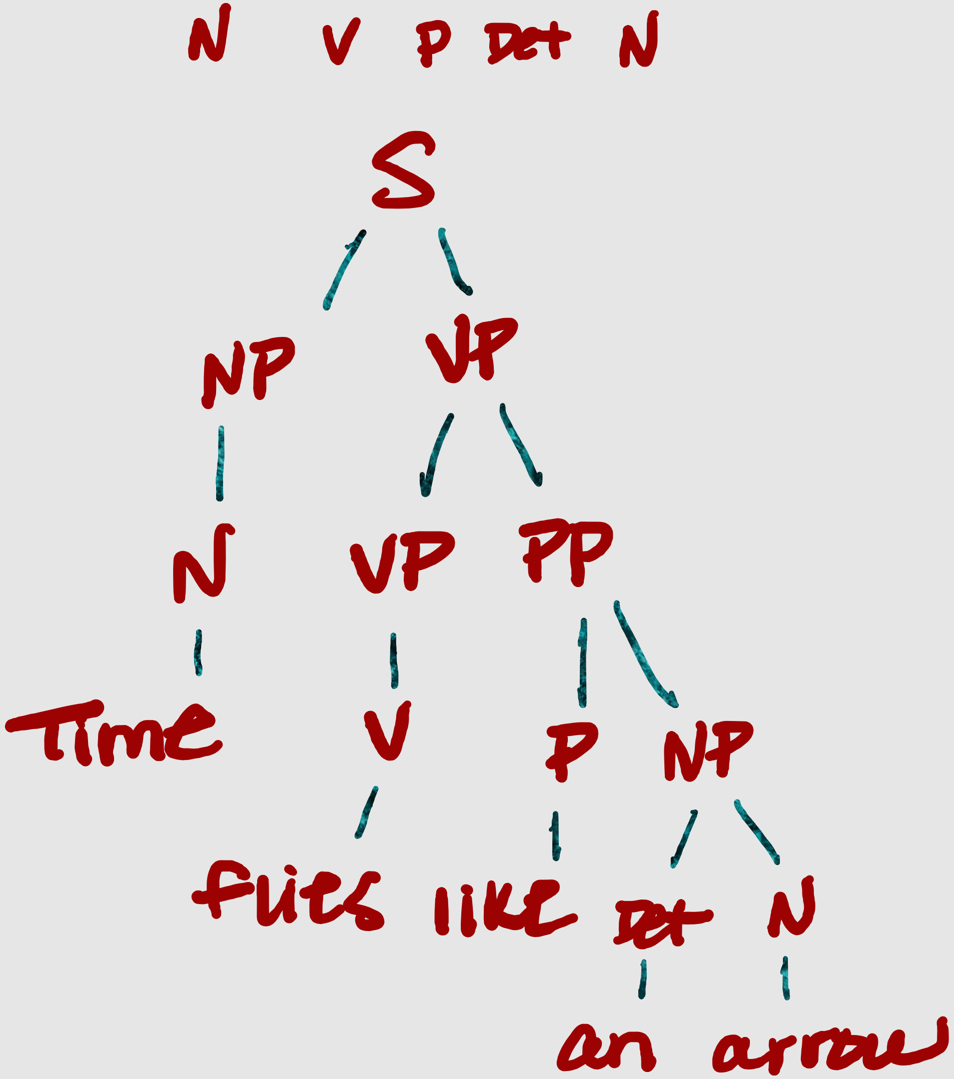
Time flies like an arrow



Case Example

Production Rules	
S ! NP VP	PP ! P NP
NP ! DET N	PP ! P
NP ! N	P ! like
NP ! N N	V ! flies | like
VP ! VP PP	DET ! a | an
VP ! V NP	N ! time | fruit | flies | arrow | banana
VP ! V	

Time flies like an arrow



CFGs and Center Embedding

Natalie knew a lot. 🙂

The zebra **that Natalie knew** knew a lot. 😕

The unicorn **that the zebra that Natalie knew knew** knew a lot. 🤯

- Formally, these sentences are all grammatical, because they can be generated by the CFG that is required for the first sentence:
  - S → NP VP
  - NP → NP RelClause
  - RelClause → that NP ate
- However, very few humans would consider the last sentence to be grammatically correct!
  - **CFGs are unable to capture bounded recursion** (e.g., embedding only one relative clause)
  - Thus, formal grammaticality isn't necessarily equivalent to human perception of grammaticality (but in this class we'll make the simplifying assumption that these are equivalent)

Refresher: Typical CFG Constituents (English)

Noun phrases (NPs)

- Simple:
  - **She** talks. (**pronoun**)
  - **Natalie** talks. (**proper noun**)
  - **A person** talks. (**determiner** + **common noun**)
- Complex:
  - **A professorial person** talks. (**determiner** + **adjective** + **common noun**)
  - **The person at the lectern** talks. (**noun phrase (determiner + common noun)** + **prepositional phrase**)
  - **The person who teaches NLP** talks. (**noun phrase (determiner + common noun)** + **relative clause**)

Visualized as production rules:

- NP → Pronoun
- NP → Proper Noun
- NP → Determiner Common Noun
- NP → Determiner Adjective Common Noun
- NP → NP PP
- NP → NP RelClause
- Pronoun → {she}
- Determiner → {a}
- Proper Noun → {Natalie}
- Common Noun → {person}
- Adjective → {professorial}

Refresher: Typical CFG Constituents (English)

Adjective Phrases (AdjP)

- AdjP → Adjective
- AdjP → Adverb AdjP
- Adj → {professorial}
- Adv → {very}
  - A very professorial person talks.

Prepositional Phrases (PP)

- PP → Preposition NP
- Preposition → {at}

Refresher: Typical CFG Constituents (English)

Verb Phrases (VPs)

- She **drinks**. (**verb**)
- She **drinks** **tea**. (**verb** + **noun phrase**)
- She **drinks tea** **from a mug**. (**verb phrase** + **prepositional phrase**)
- Visualized as production rules:
  - VP → V
  - VP → V NP
  - VP → V NP PP
  - VP → VP PP
  - V → {drinks}

We can also capture subcategorization this way!

- She **drinks**. (**verb**)
- She **drinks** **tea**. (**verb** + **noun phrase**)
- She **gives** **him** **tea**. (**verb phrase** + **noun phrase** + **noun phrase**)
- Visualized as production rules:
  - VP → $V_{intransitive}$
  - VP → $V_{transitive}$ NP
  - VP → $V_{ditransitive}$ NP NP
  - $V_{intransitive}$ → {drinks, talks}
  - $V_{transitive}$ → {drinks}
  - $V_{ditransitive}$ → {gives}

To comprehensively cover English grammar, more complex production rules are necessary.

- We want to prevent against grammatical incorrectness:
  - She drinks tea. 🙂
  - I drinks tea. 🤨
  - They drinks tea. 🤨
- We can do this by establishing different production rules for different tenses or other phenomena:
  - Present Tense: She drinks tea.
  - Simple Past Tense: She drank tea.
  - Past Perfect Tense: She has drunk tea.
  - Future Perfect Tense: She will have drunk tea.
  - Passive: The tea was drunk by her.
  - Progressive: She will be drinking tea.

- $VP \rightarrow V_{have}\ VP_{pastPart}$
- $VP \rightarrow V_{be}\ VP_{pass}$
- $VP_{pastPart} \rightarrow V_{pastPart}\ NP$
- $VP_{pass} \rightarrow V_{pastPart}\ PP$
- $V_{have} \rightarrow \{has\}$
- $V_{pastPart} \rightarrow \{drunk\}$
- etc.…

Refresher: Typical CFG Constituents (English)

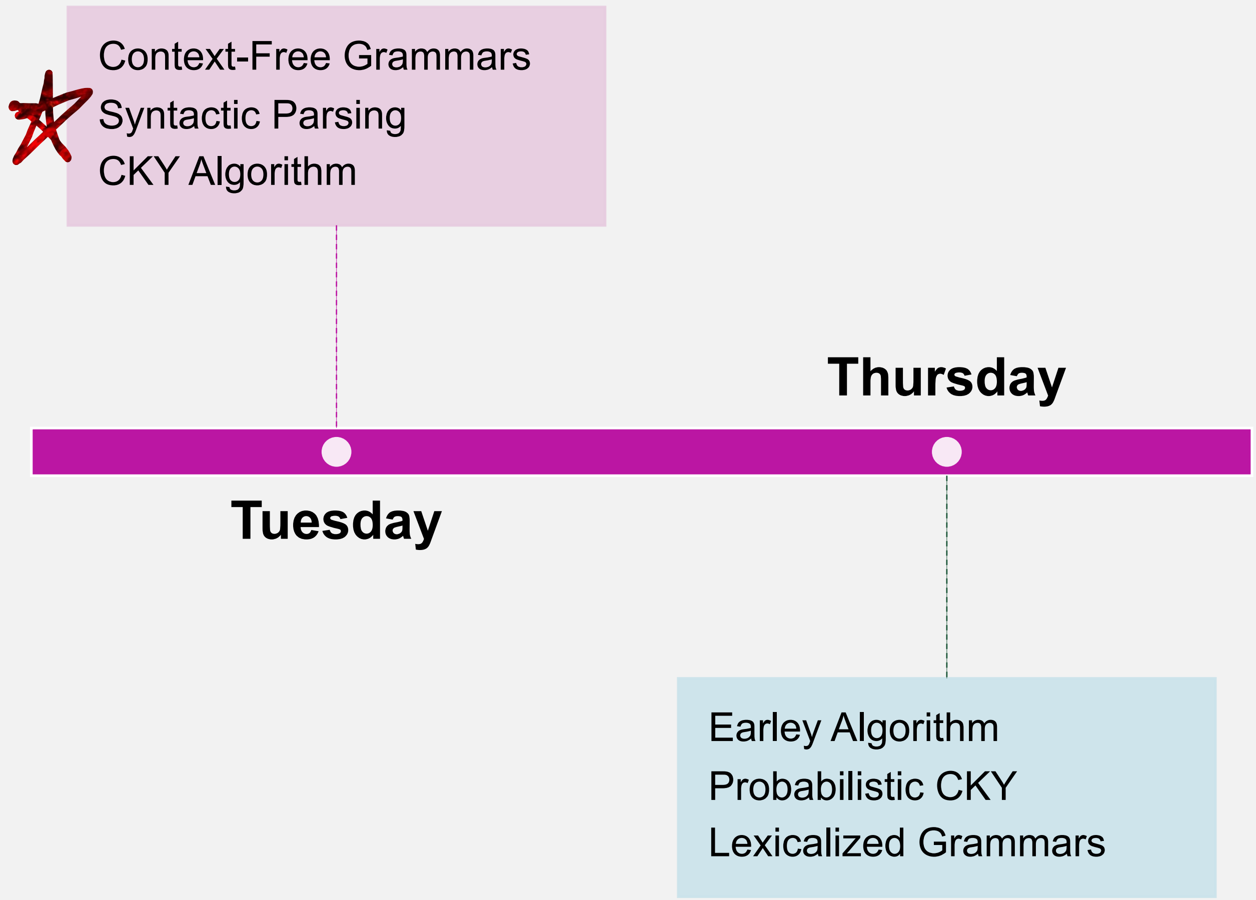
- Production rules can also recursively include sentences
  - She drinks tea. (noun phrase + verb phrase)
  - Sometimes, she drinks tea. (adverbial phrase + sentence)
  - In England, she drinks tea. (prepositional phrase + sentence)
- Visualized as production rules:
  - S → NP VP
  - S → AdvP S
  - S → PP S
- And they can cover questions:
  - Yes/No Questions
    - Auxiliary + Subject + Verb Phrase
      - Does she drink tea?
    - YesNoQ → Aux NP VP
  - Wh-Questions
    - Subject wh-questions contain a wh-word, an auxiliary, and a verb phrase
      - Who has drunk the tea?
    - Object wh-questions contain a wh-word, an auxiliary, a noun phrase and a verb phrase
      - What does Natalie drink?

Coordinating Conjunctions and Relative Clauses

- **She drinks tea** and **he drinks coffee**.
- **Natalie** and **her mom** drink tea.
- She **drinks tea** and **eats cake**.
- Production Rules:
  - S → S conj S
  - NP → NP conj NP
  - VP → VP conj VP

- **Relative clauses modify a noun phrase** by adding extra information
  - Rather than having their own noun phrase, it is understood that the NP is filled by the NP that the relative clause modifies
  - She had **a poodle** **that drank my tea**. → that = a poodle
- There are two types of relative clauses
  - Subject: She had a poodle **that drank my tea**.
    - We cannot drop the relative pronoun
  - Object: I'd really been enjoying the tea **that her poodle drank**.
    - We can drop the relative pronoun and the sentence still works

This Week's Topics



CFGs and dependency grammars for regular languages can be highly complex!

However, they facilitate automated syntactic and semantic parsing, which helps us better understand language

Syntactic parsing: The process of automatically recognizing and assigning syntactic (grammatical) roles to the constituents within sentences

Why is syntactic parsing useful?

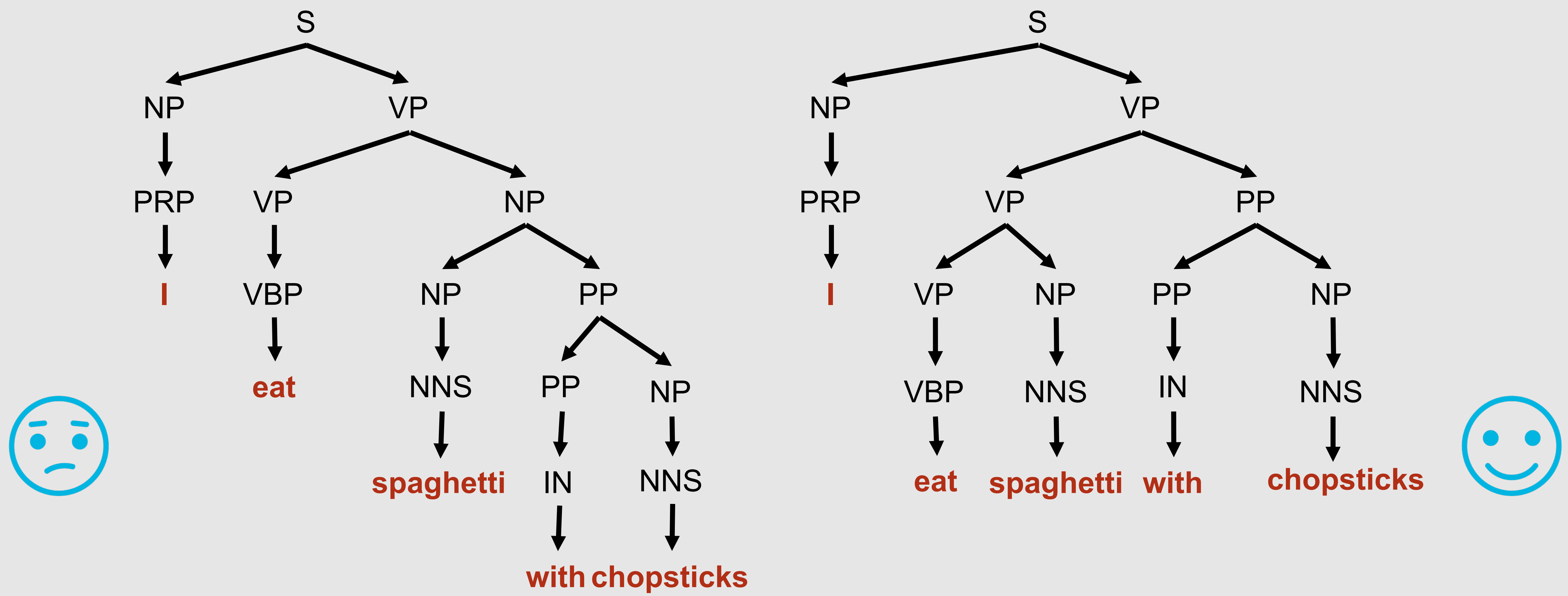
- Lots of reasons! For example:
  - Grammar checking
  - Downstream applications
    - Question answering
    - Information extraction

What courses were **taught by UIC CS assistant professors in 2023**?

Subject = courses …don't return a list of UIC CS assistant professors!

Recognition vs. Parsing

- **Recognition:** Deciding whether a sentence belongs to the language specified by a formal grammar.
- **Parsing:** Producing a parse tree for the sentence based on that formal grammar.
- Both tasks are necessary for generating correct syntactic parses!
  - Failure to accurately recognize whether a sentence can be parsed will lead to **misparses**, which will in turn lead to additional errors in downstream applications.
- Parsing is more "difficult" (greater time complexity) than recognition



Remember, language is ambiguous!

Input sentences may have many possible parses

There are also many ways to generate parse trees.

Top-Down Parsing:	Bottom-Up Parsing:
Goal-driven Builds parse tree from the start symbol down to the terminal nodes	Data-driven Builds parse tree from the terminal nodes up to the start symbol

Top-Down Parsing

- Assume that the input can be derived by the designated start symbol ***S***
- Find the tops of all trees that can start with ***S***
  - Look for all production rules with ***S*** on the left-hand side
- Find the tops of all trees that can start with those constituents
- (Repeat recursively until terminal nodes are reached)
- Trees whose leaves fail to match all words in the input sentence can be rejected, leaving behind trees that represent successful parses

Top-Down Parsing: Example

Input Sentence:

Book that flight.

Grammar:

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon:

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

Top-Down Parsing: Example

Book that flight.

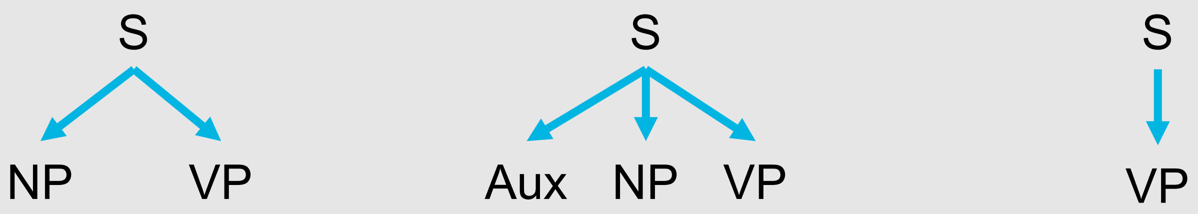
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

S S S

Top-Down Parsing: Example

Book that flight.

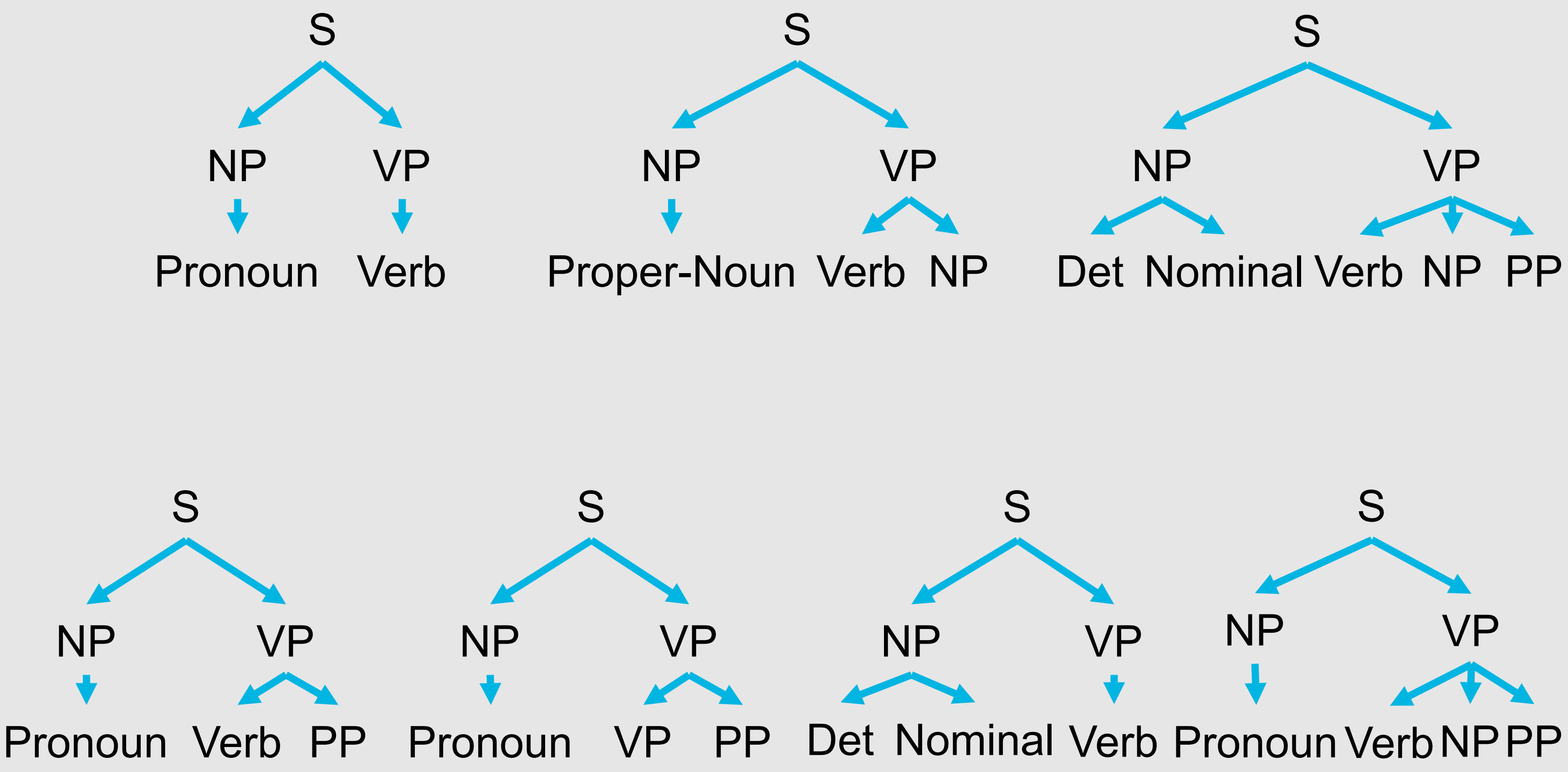
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

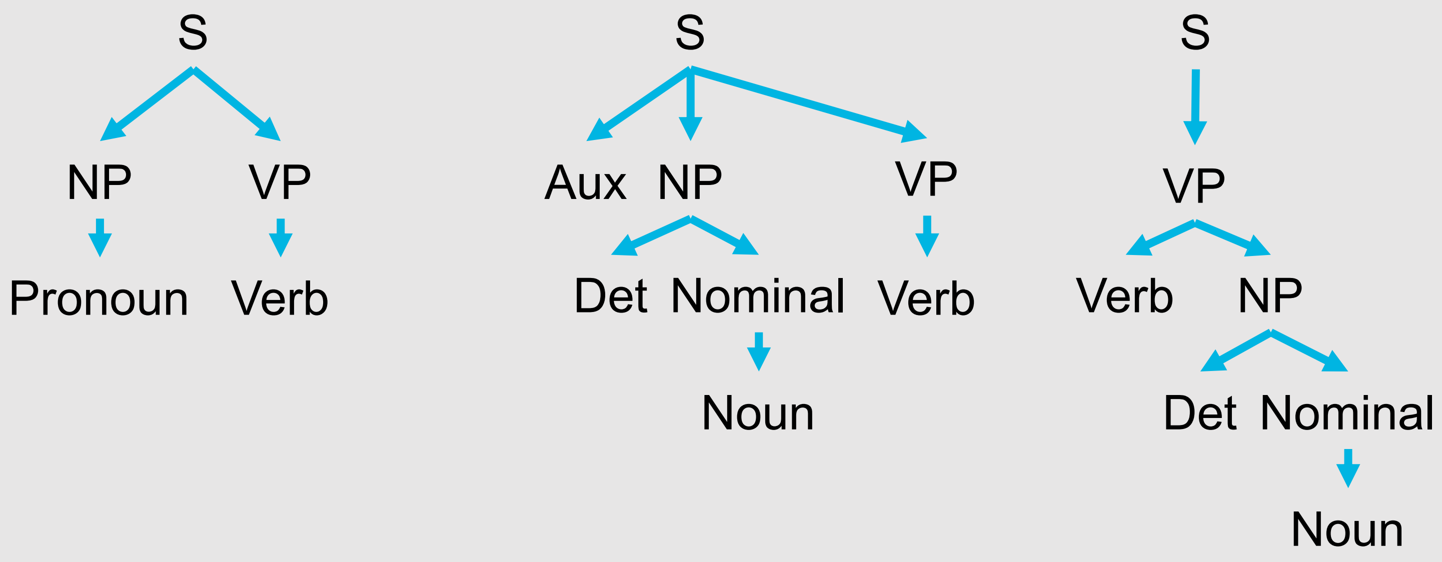


…and many more!

Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

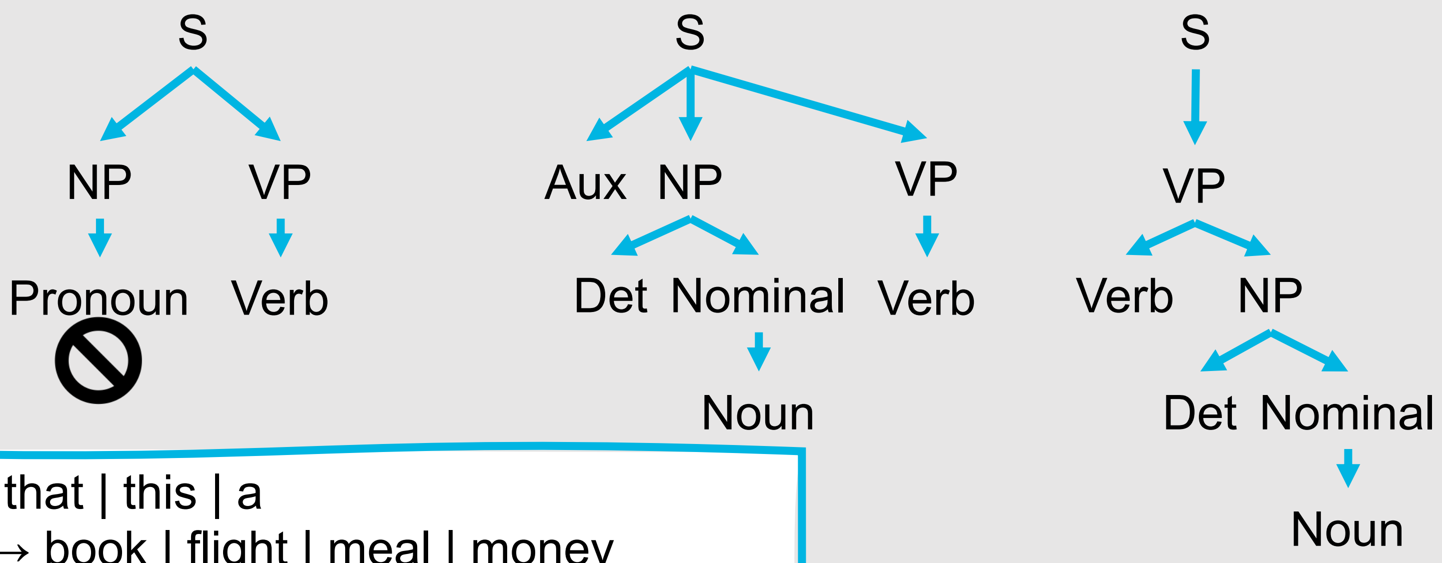


…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



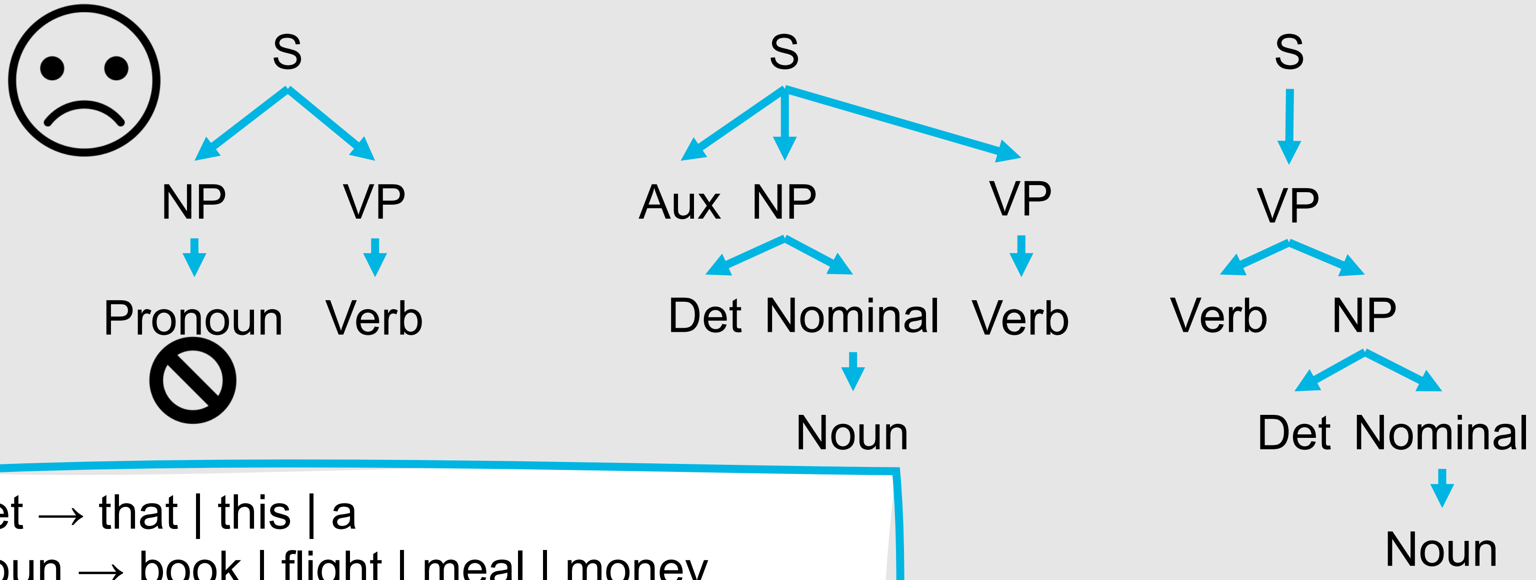
Det → that | this | a
Noun → book | flight | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



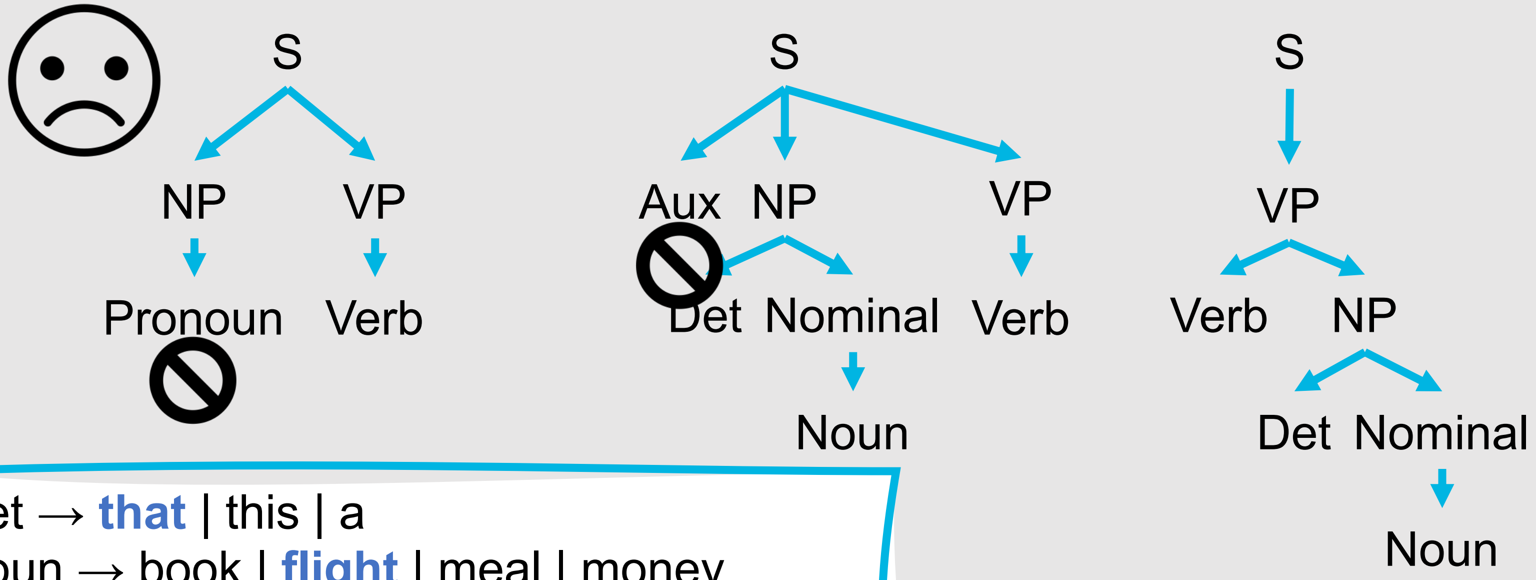
Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



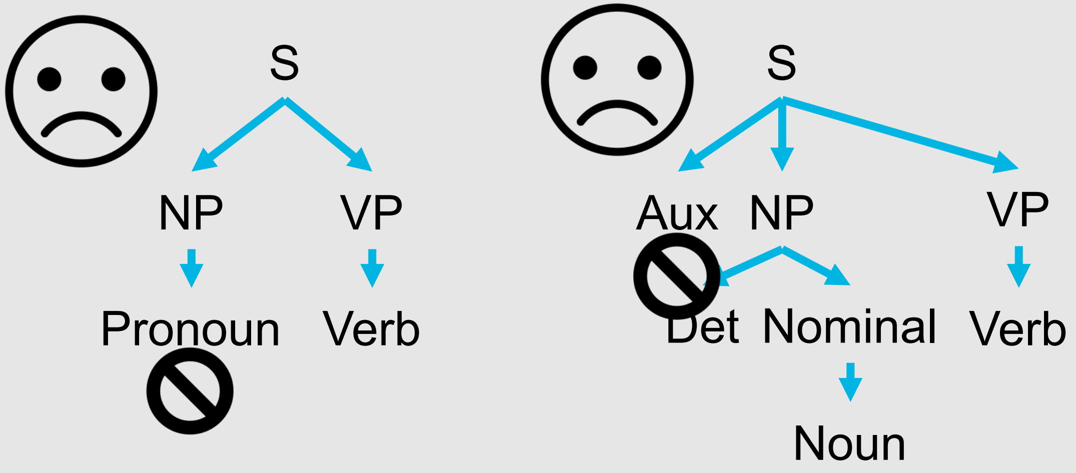
Det → **that** | this | a
Noun → book | **flight** | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

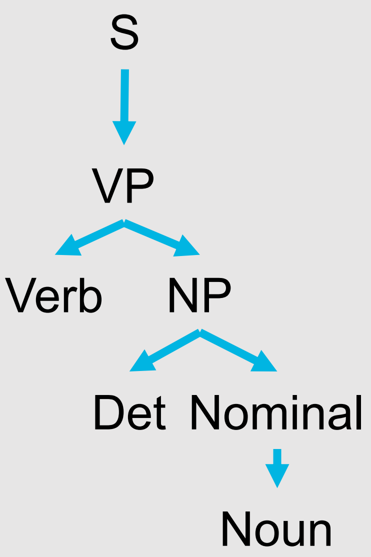
…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP





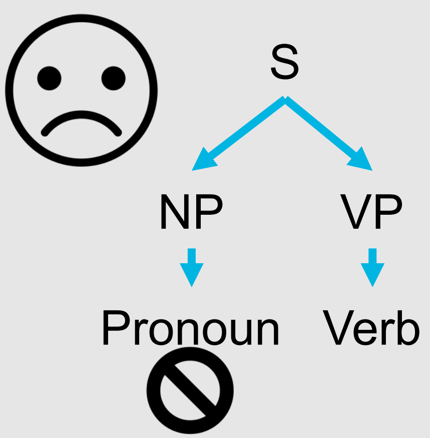
Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

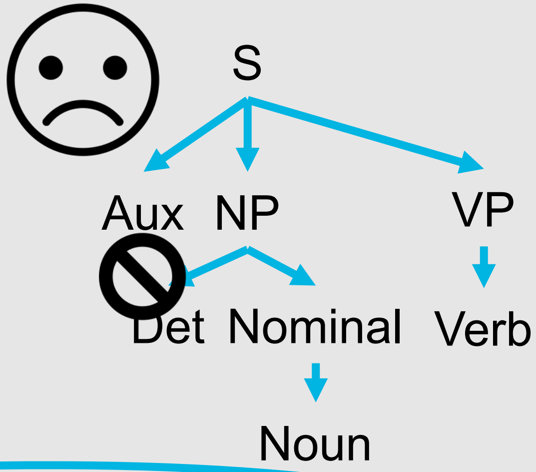
…and many, many more not shown!

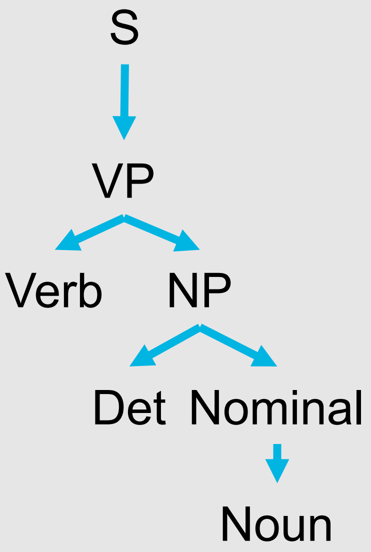
Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP







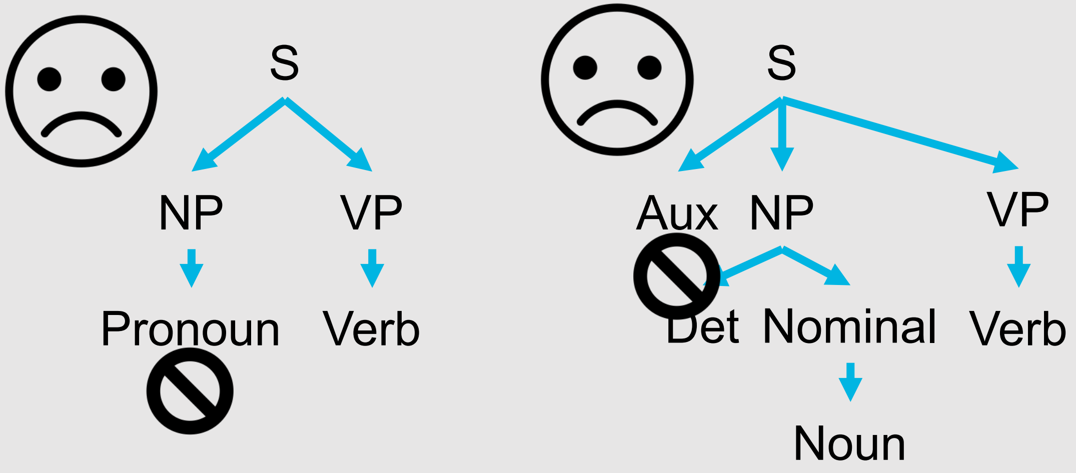
Det → **that** | this | a
Noun → book | **flight** | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

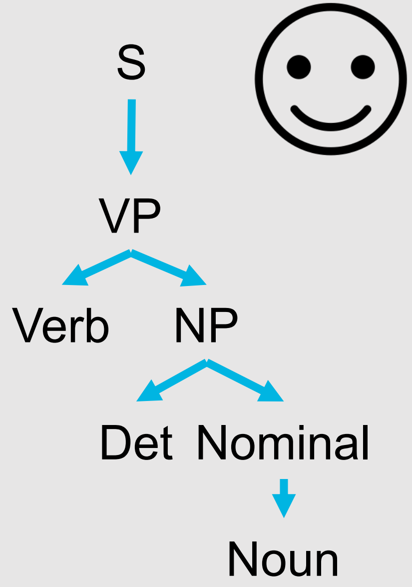
…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP





Det → **that** | this | a
Noun → book | **flight** | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

…and many, many more not shown!

Bottom-Up Parsing

- Earliest known parsing algorithm!
- Starts with the words in the input sentence, and tries to build trees from those words up by applying rules from the grammar one at a time
  - Looks for places in the in-progress parse where the righthand side of a production rule might fit
- Success = parser builds a tree rooted in the start symbol ***S*** that covers all of the input words

Bottom-Up Parsing: Example

Input Sentence:

Book that flight.

Grammar:

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon:

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

Bottom-Up Parsing: Example

Book that flight.

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

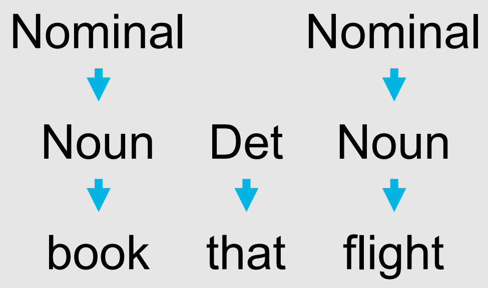
Noun	Det	Noun
↓	↓	↓
book	that	flight

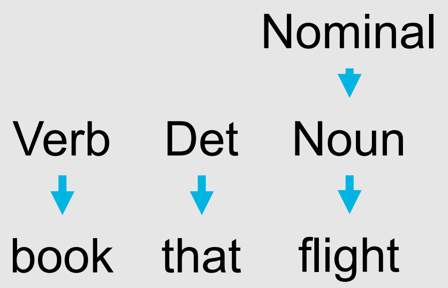
Verb	Det	Noun
↓	↓	↓
book	that	flight

Bottom-Up Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

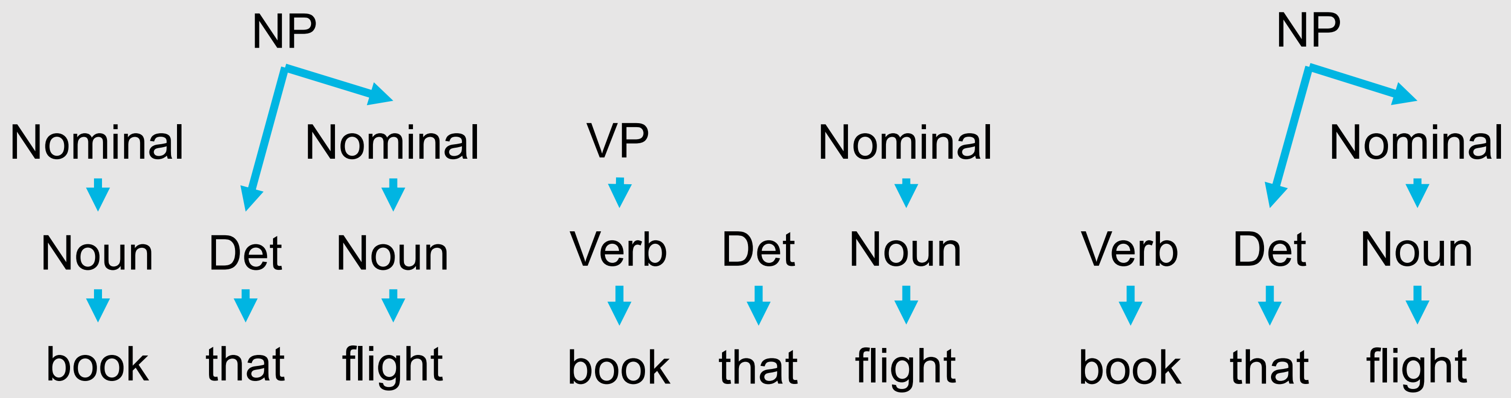




Bottom-Up Parsing: Example

Book that flight.

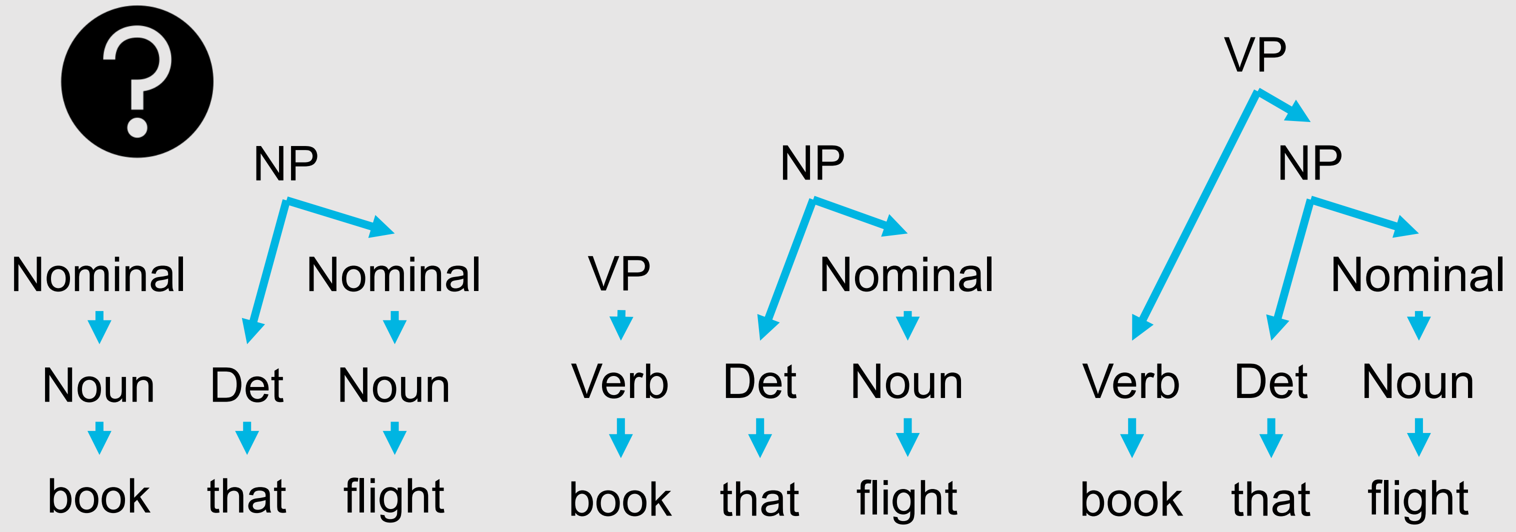
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



Bottom-Up Parsing: Example

Book that flight.

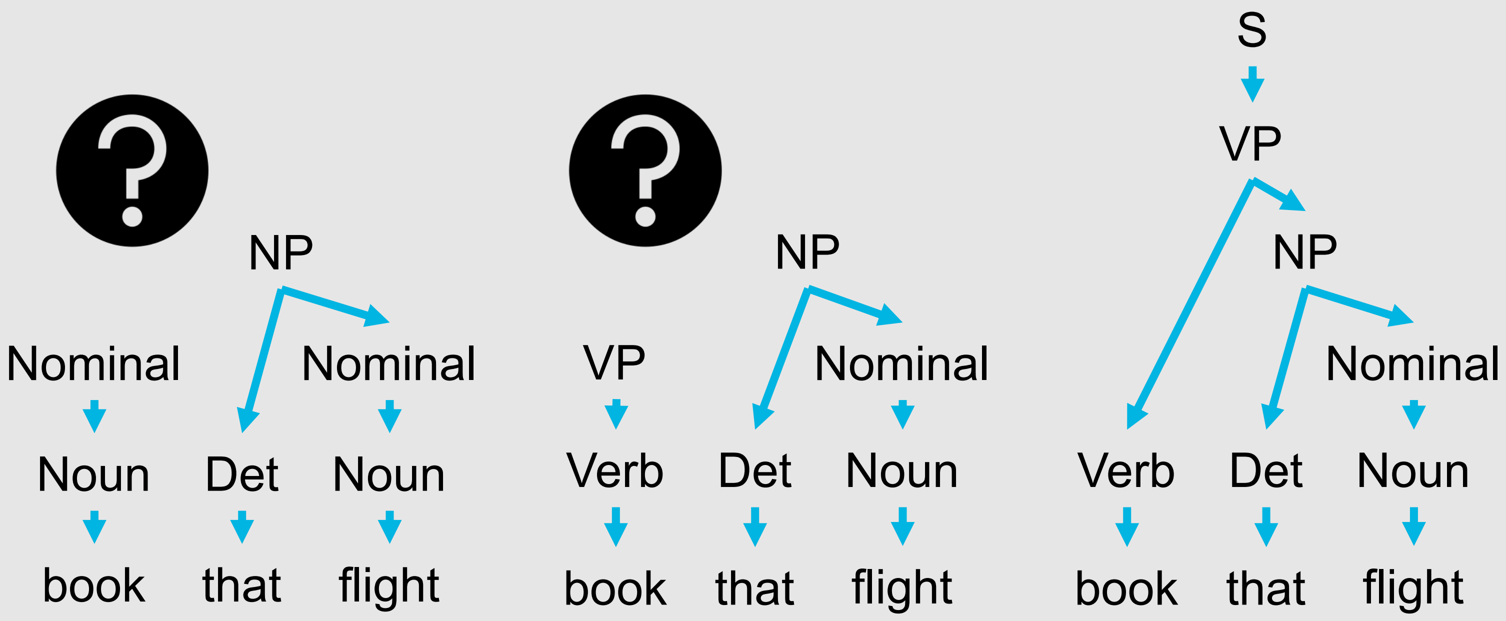
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



Bottom-Up Parsing: Example

Book that flight.

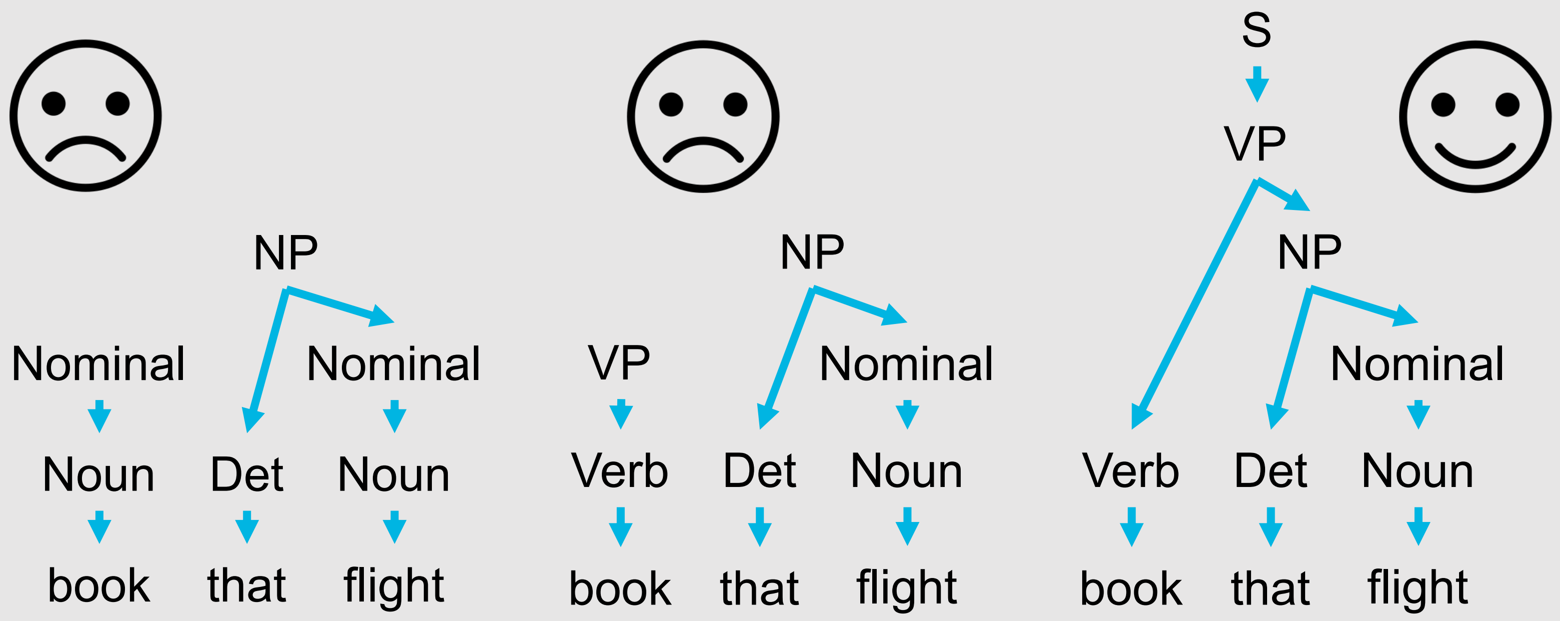
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



Bottom-Up Parsing: Example

Book that flight.

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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



Top-Down vs. Bottom-Up Parsing

Top-Down Parsing

- Pros:
  - Never wastes time exploring invalid trees
- Cons:
  - Spends considerable effort on trees that are not consistent with the input

Bottom-Up Parsing

- Pros:
  - Never suggests trees that are inconsistent with the input
- Cons:
  - Generates many trees and subtrees that cannot result in a valid sentence (according to production rules specified by the grammar)

This Week's Topics

Tuesday

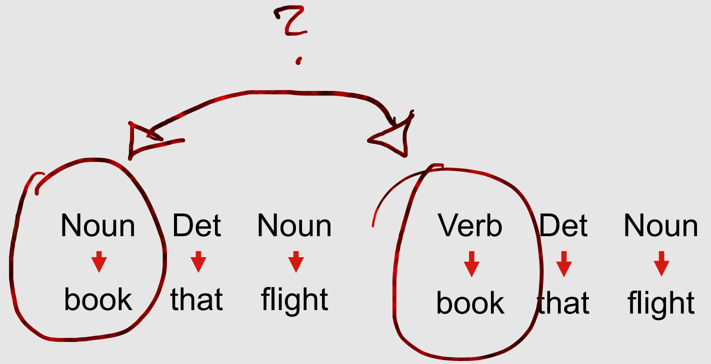
- Context-Free Grammars
- Syntactic Parsing
- CKY Algorithm

Thursday

- Earley Algorithm
- Probabilistic CKY
- Lexicalized Grammars

Many forms of ambiguity can arise during syntactic parsing!

- **Structural Ambiguity:** Occurs when a grammar allows for more than one possible parse for a given sentence
  - **Attachment Ambiguity:** Occurs when a constituent can be attached to a parse tree at more than one place
    - I eat spaghetti *with chopsticks.*
  - **Coordination Ambiguity:** Occurs when different sets of phrases can be conjoined by a conjunction
    - I grabbed a muffin from the table marked "nut-free scones *and* muffins," hoping I'd parsed the sign correctly.
- **Local Ambiguity:** Occurs when a word may be interpreted multiple ways



- Det → that | this | a
- Noun → **book** | flight | meal | money
- Verb → **book** | include | prefer
- Pronoun → I | she | me
- Proper-Noun → Houston | NWA
- Aux → does
- Preposition → from | to | on | near | through

All of this ambiguity can lead to really complex search spaces.

- **Backtracking** approaches expand the search space incrementally, systematically exploring one state at a time
  - When they arrive at trees inconsistent with the input, they return to an unexplored alternative
  - However, in doing so they tend to discard valid subtrees …this means that time-consuming work needs to be repeated
- More efficient approach?
  - **Dynamic programming**

Dynamic Programming Parsing Methods

- Widely used methods:
  - Cocke-Kasami-Younger (**CKY**) algorithm
  - **Earley** algorithm

CKY Algorithm

- One of the earliest recognition and parsing algorithms
- **Bottom-up dynamic programming**
- Standard version can only recognize CFGs in **Chomsky Normal Form** (CNF)

Chomsky Normal Form

- Grammars are restricted to production rules of the form:
  - $A \rightarrow B\ C$
  - $A \rightarrow w$
- This means that the righthand side of each rule must expand to either two non-terminals or a single terminal
- Any CFG can be converted to a corresponding CNF grammar that accepts exactly the same set of strings as the original grammar!

How does this conversion work?

- Three situations we need to address:
  1. Production rules that mix terminals and non-terminals on the righthand side
  2. Production rules that have a single non-terminal on the righthand side (**unit productions**)
  3. Production rules that have more than two non-terminals on the righthand side
- Situation #1: **Introduce a dummy non-terminal that covers only the original terminal**
  - INF-VP → to VP could be replaced with INF-VP → TO VP and TO → to
- Situation #2: **Replace the non-terminals with the non-unit production rules to which they eventually lead**
  - A → B and B → w could be replaced with A → w
- Situation #3: **Introduce new non-terminals that spread longer sequences over multiple rules**
  - A → B C D could be replaced with A → B X1 and X1 → C D

Original	CNF
S → NP VP	S → NP VP
S → AdjP NP VP	S → X1 VP
	X1 → AdjP NP
S → VP	S → book | include | prefer

CKY Algorithm

- With the grammar in CNF, each non-terminal node above the POS level of the parse tree will have exactly two children
- Thus, a two-dimensional matrix can be used to encode the tree structure
- Each cell [*i*,*j*] contains a set of non-terminals that represent all constituents spanning positions *i* through *j* of the input
  - Cell that represents the entire input resides in position [0,*n*]

CKY Algorithm

- Non-terminal entries: For each constituent [*i*,*j*], there is a position, *k*, where the constituent can be split into two parts such that $i < k < j$
  - [*i*,*k*] must lie to the left of [*i*,*j*] somewhere along row *i*, and [*k*,*j*] must lie beneath it along column *j*
- To fill in the parse table, we proceed in a bottom-up fashion so when we fill a cell [*i*,*j*], the cells containing the parts that could contribute to this entry have already been filled

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0					
1					
2					
3					
4					

CKY Algorithm: Example

Det → that | this | a | the
Noun → **book** | flight | meal | money
Verb → **book** | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → **book** | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → **book** | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → **book** | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1					
2					
3					
4					

CKY Algorithm: Example

Det → that | this | a | **the**
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1		Det			
2					
3					
4					

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | **flight** | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | **flight** | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1		Det			
2			Noun, Nominal		
3					
4					

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | **through**

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1		Det			
2			Noun, Nominal		
3				Prep.	
4					

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

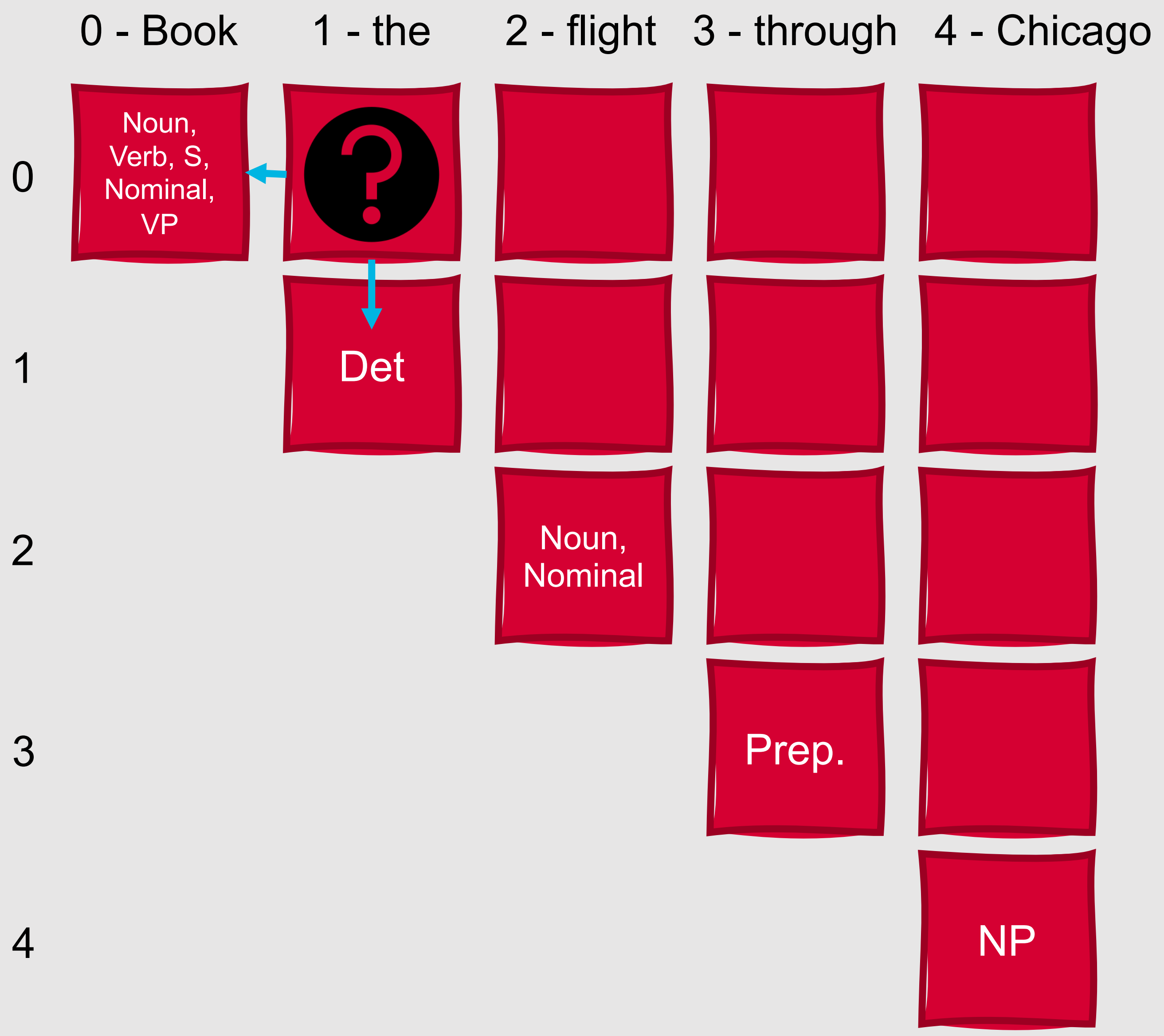
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → **Chicago** | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1		Det			
2			Noun, Nominal		
3				Prep.	
4					NP

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

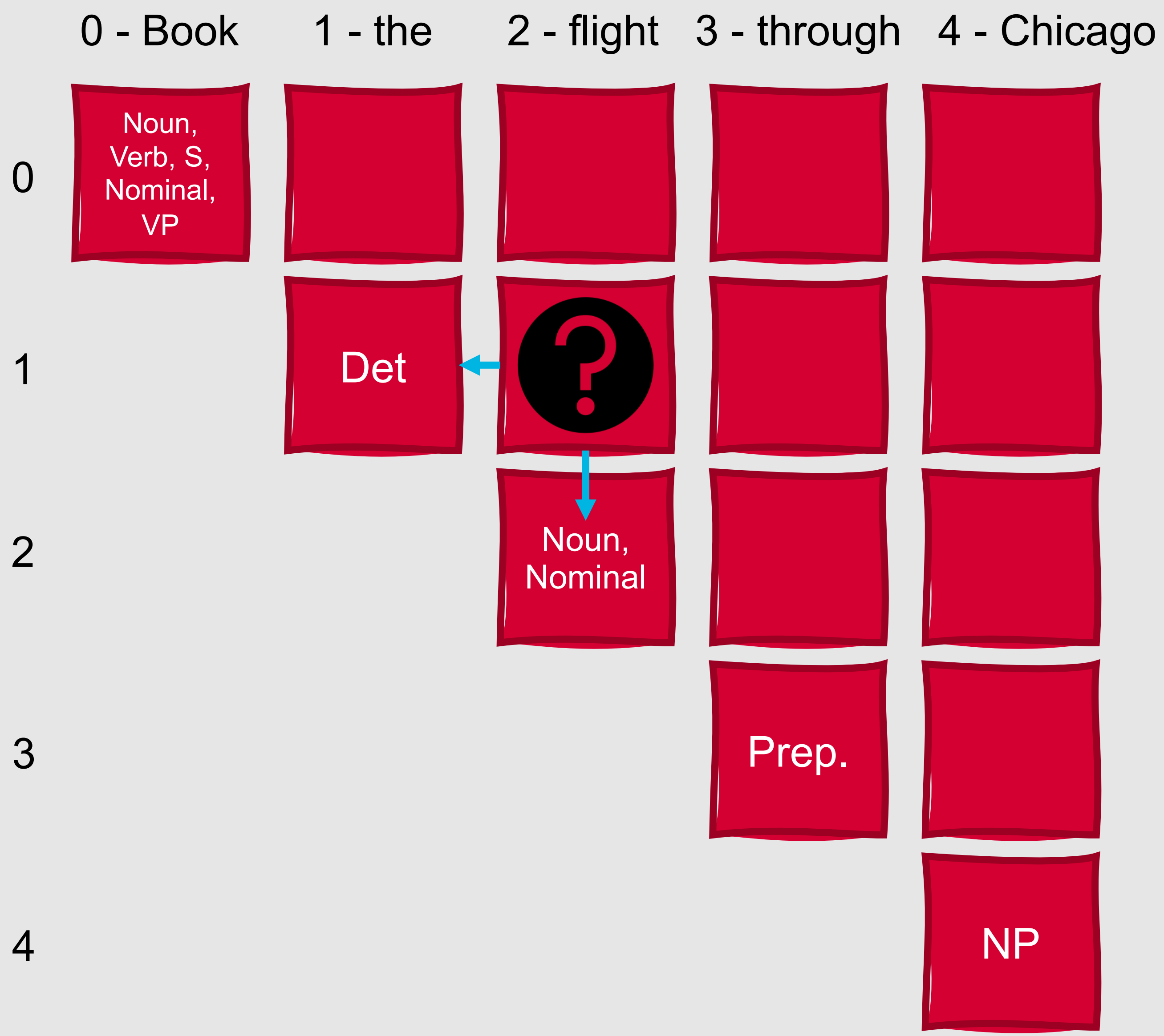
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

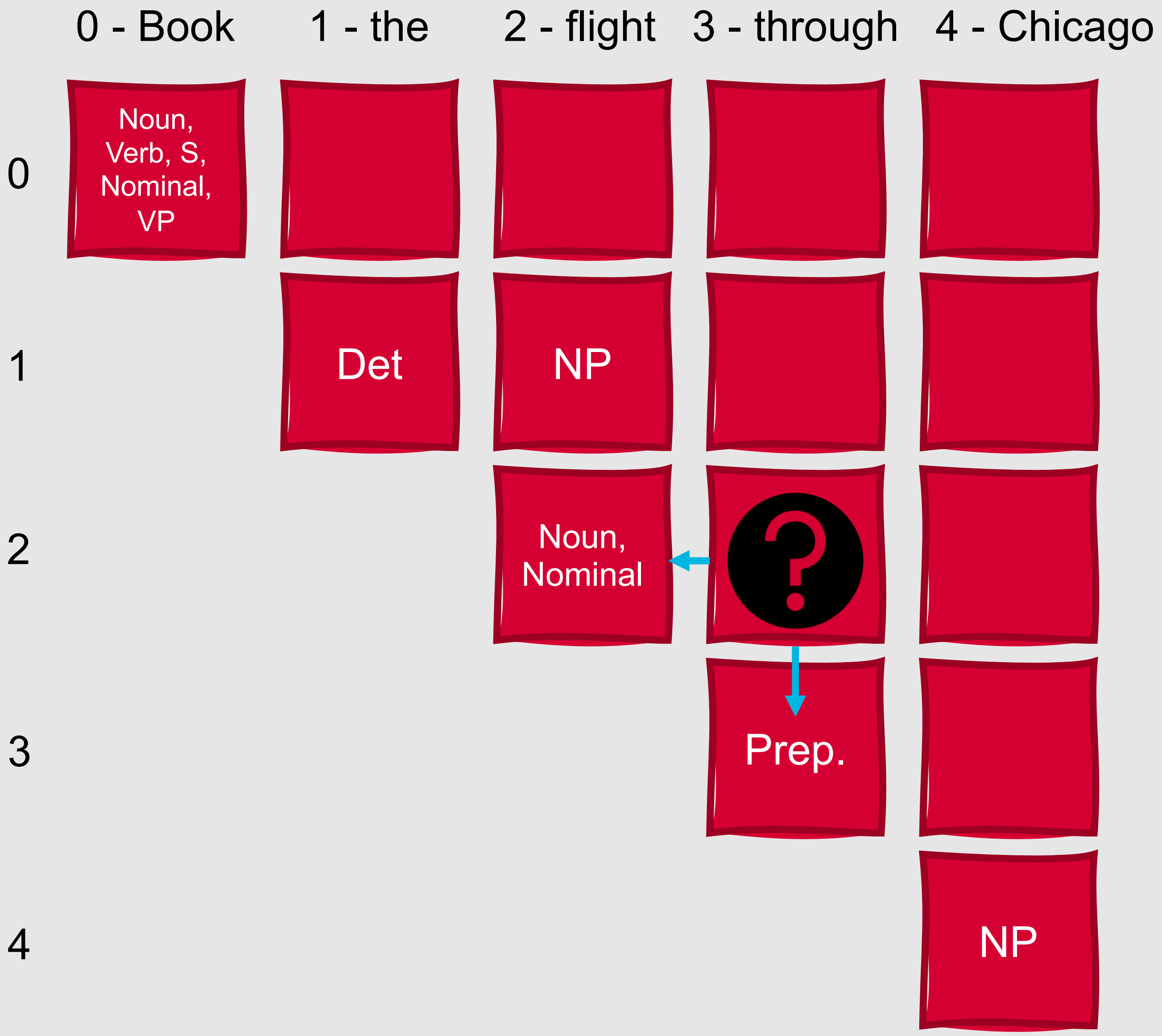
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1		Det	NP		
2			Noun, Nominal		
3				Prep.	
4					NP

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1		Det	NP		
2			Noun, Nominal		
3				Prep.	?
4					NP

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

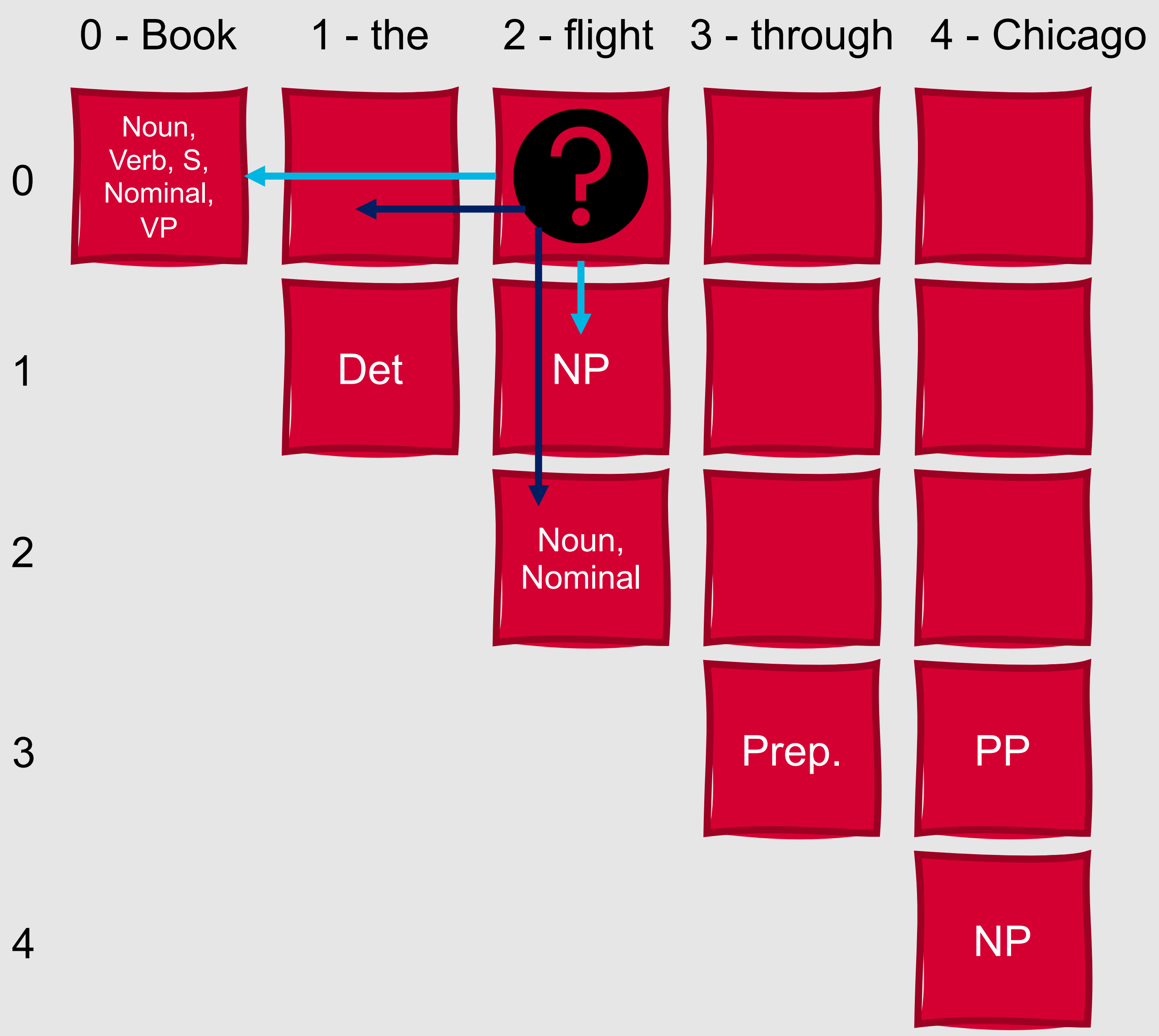
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP				
1		Det	NP		
2			Noun, Nominal		
3				Prep.	PP
4					NP

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

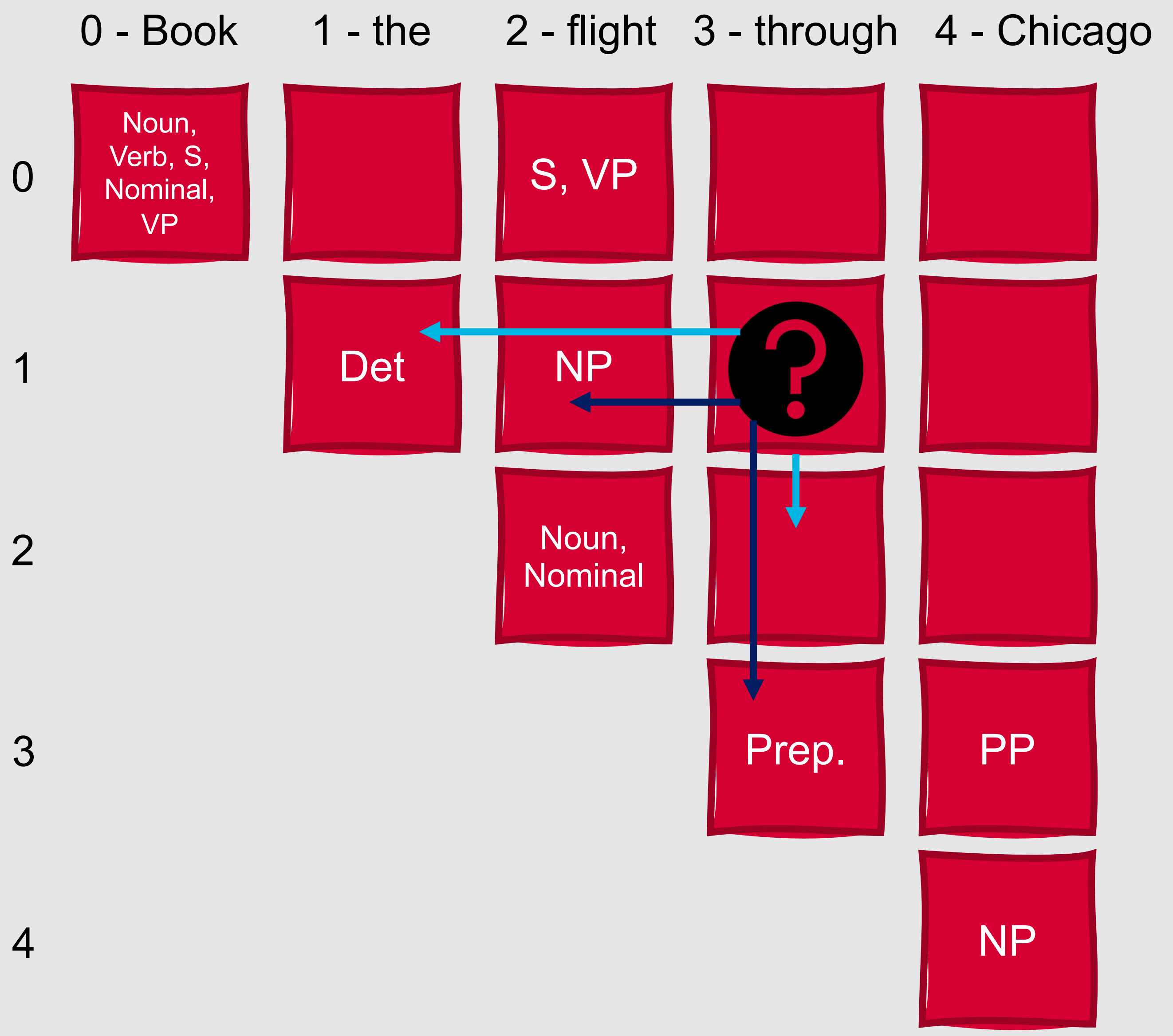
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP		S, VP		
1		Det	NP		
2			Noun, Nominal		
3				Prep.	PP
4					NP

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

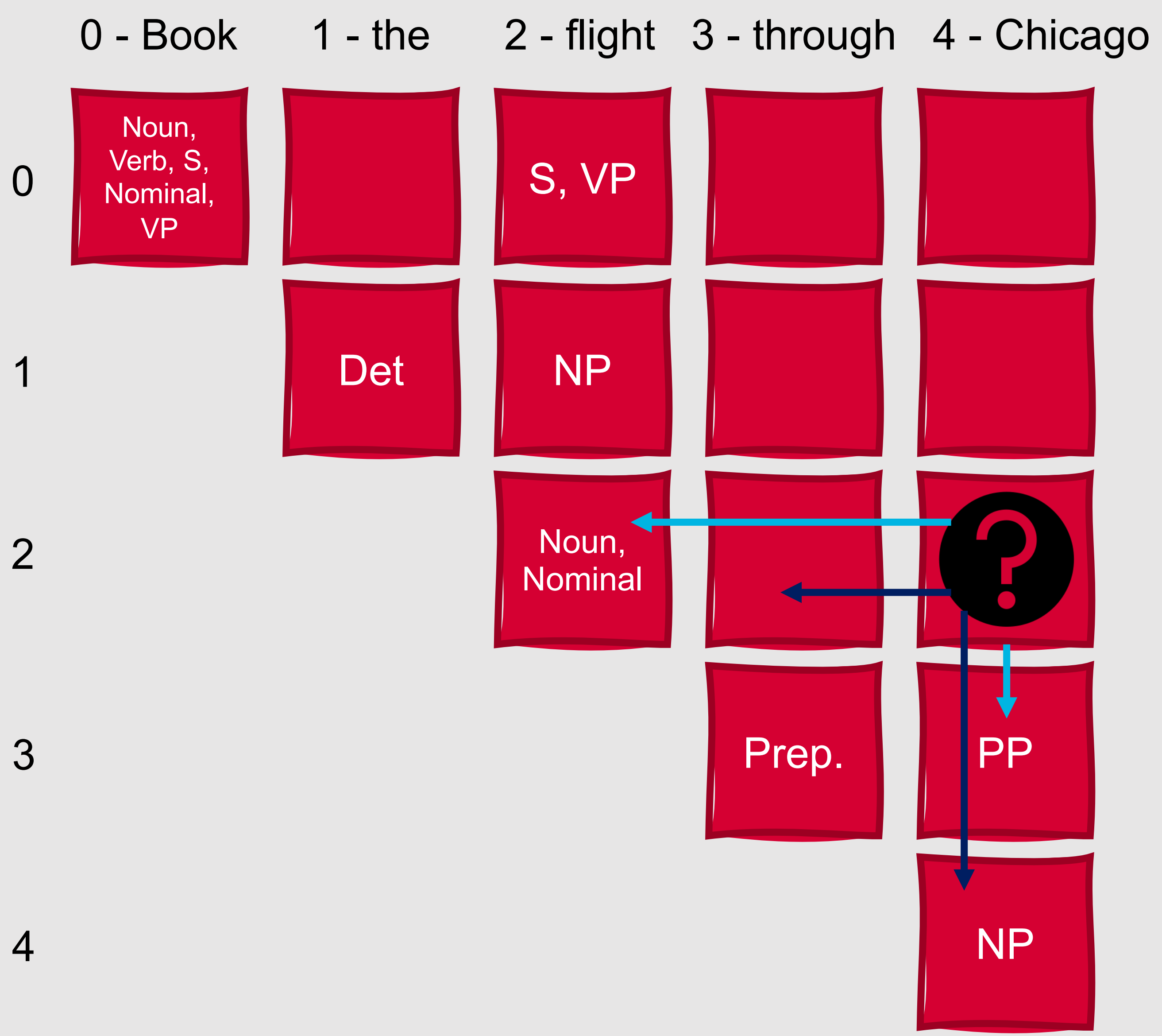
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

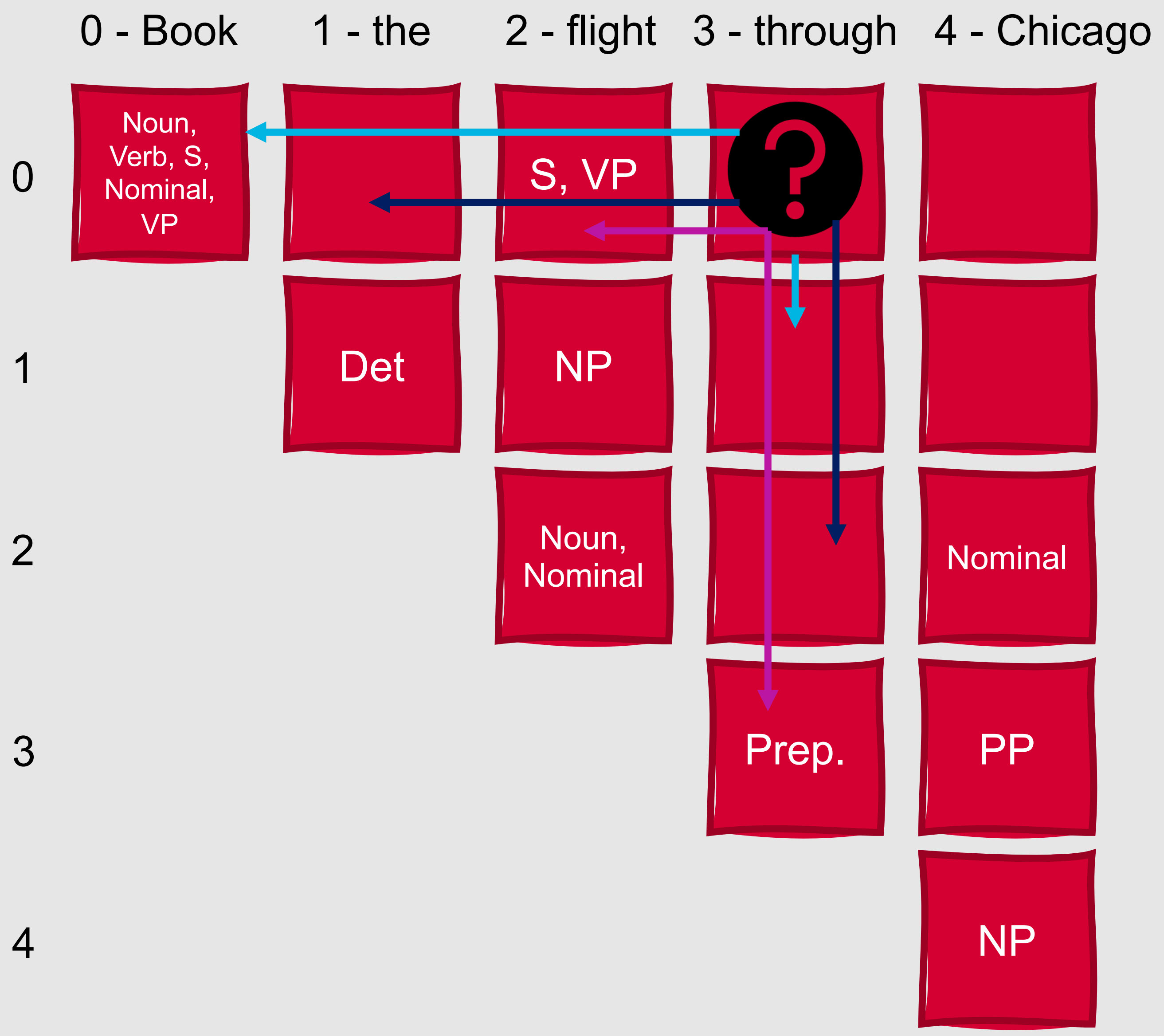
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP		S, VP		
1		Det	NP		
2			Noun, Nominal		Nominal
3				Prep.	PP
4					NP

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

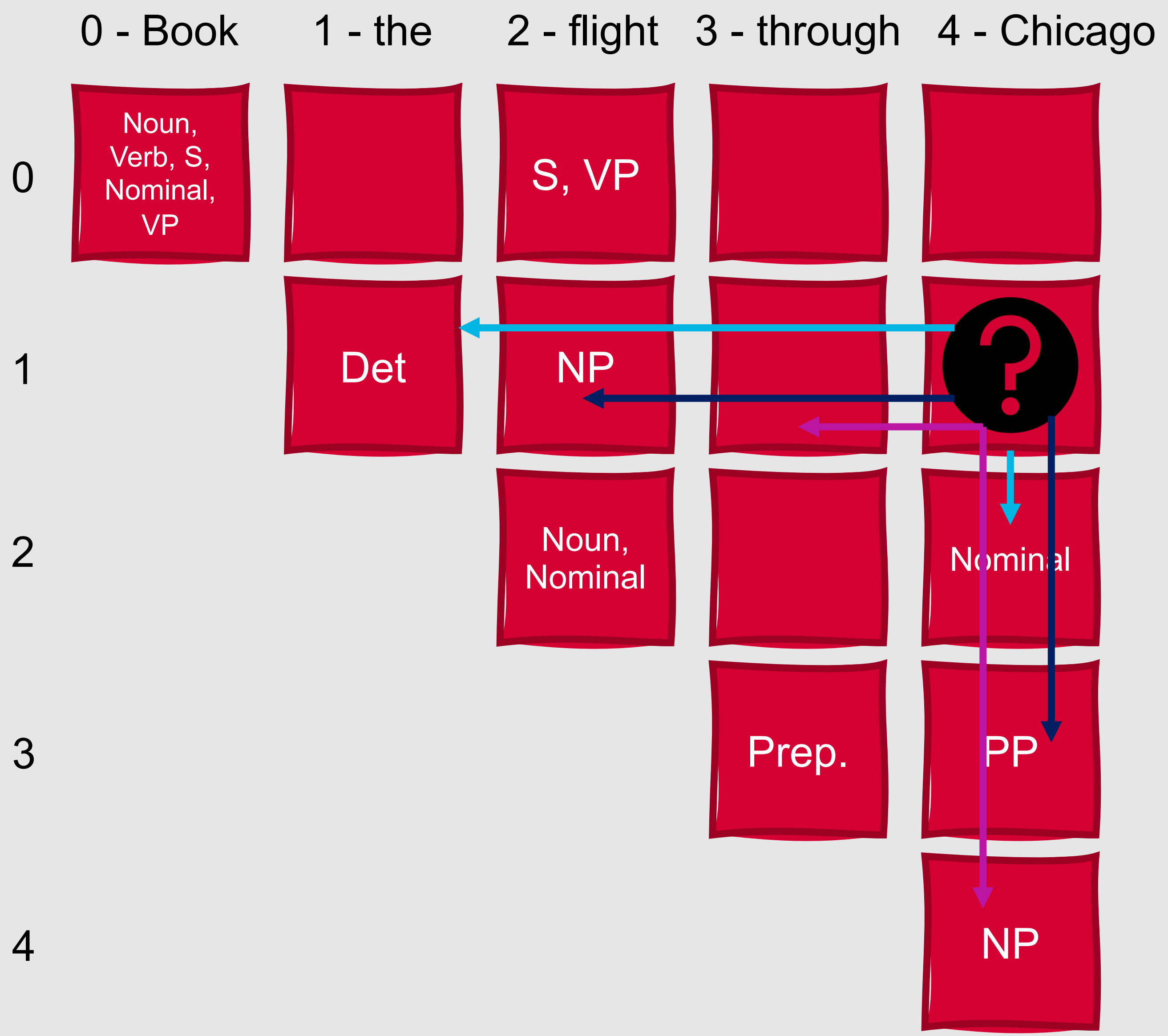
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

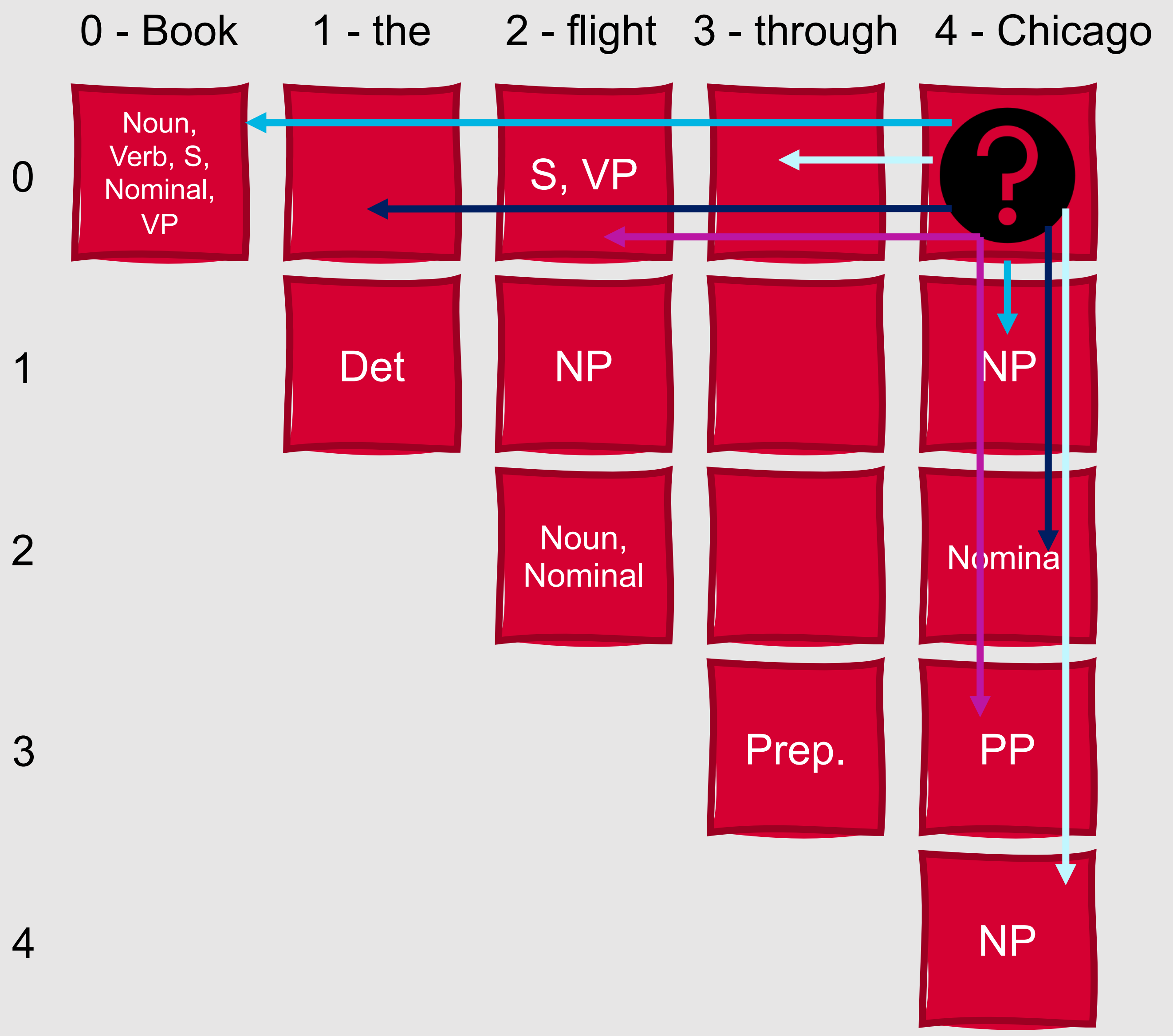
S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP		S, VP		
1		Det	NP		NP
2			Noun, Nominal		Nominal
3				Prep.	PP
4					NP

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

	0 - Book	1 - the	2 - flight	3 - through	4 - Chicago
0	Noun, Verb, S, Nominal, VP		S, VP		S, VP_1, VP_2
1		Det	NP		NP
2			Noun, Nominal		Nominal
3				Prep.	PP
4					NP

If we arrive at VP in multiple ways, each way is an alternative parse (VP_1, VP_2, …, VP_n).

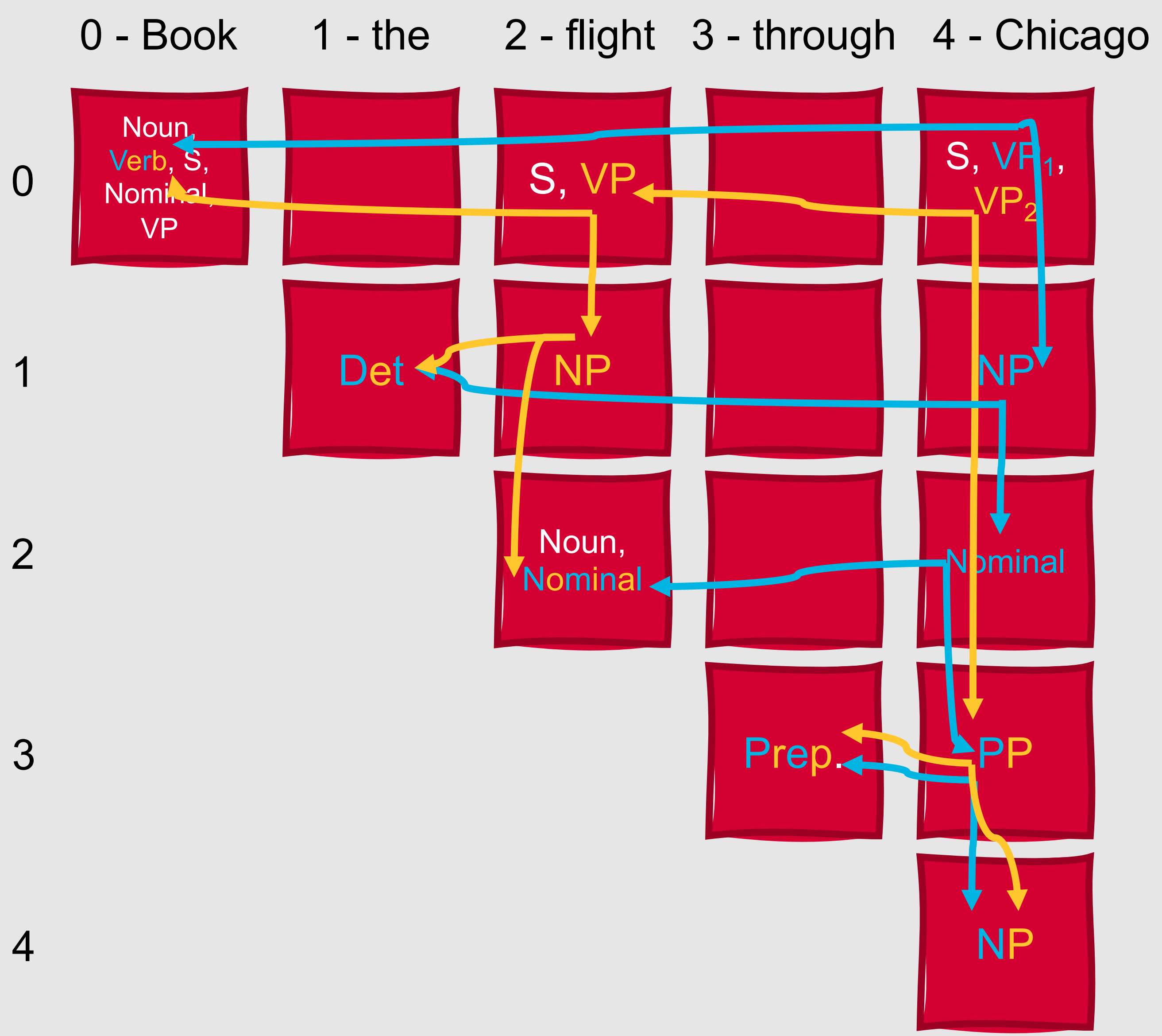
CKY Algorithm

- In the previous example, we **recognized** a valid that this sentence was valid according to our grammar by finding an S in cell [0,n]
- To return all possible parses, we need to also pair each non-terminal with pointers to the table entries from which it was derived
- Then, we can choose a non-terminal and recursively retrieve its component constituents from the table
- Complexity of this algorithm:
  - Time complexity: $O(n^3)$
  - Space complexity: $O(n^2)$

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Preposition → from | to | on | near | through

S → NP VP
S → book | include | prefer
S → Verb NP
NP → I | she | me
NP → Chicago | Dallas
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



Summary: Constituency Grammars

Constituency grammars describe a language's syntactic structure

Constituents, a core component of constituency grammars, are groups of words that function as a single unit

There are many ways to represent constituency grammars, but the most common way is by using **trees**

Constituency grammars can generate any sentences belonging to their language using (potentially recursive) combinations of **production rules**

Syntactic parsing is a way to automatically describe the structure of an input sentence according to a constituency grammar

Syntactic parsing can be performed using either a **top-down** or a **bottom-up approach**

One popular bottom-up approach is the **CKY** algorithm

This Week's Topics

Context-Free Grammars
Syntactic Parsing
CKY Algorithm

Tuesday

Thursday

Earley Algorithm
Probabilistic CKY
Lexicalized Grammars

Earley Parsing

- Top-down dynamic parsing approach
- Table is length *n*+1, where *n* is equivalent to the number of words
- Table entries contain three types of information:
  - A single grammar rule
  - Information about the progress made in completing that rule
    - A • within the righthand side of a state's grammar rule indicates the progress made towards recognizing it
  - The position of the in-progress rule with respect to the input
    - Represented by two numbers, indicating (1) where the state begins, and (2) where its dot lies

Example States

- Input: Book that flight.
- S → • VP, [0,0]
  - Top-down prediction for this particular kind of S
  - First 0: Constituent predicted by this state should begin at the start of the input
  - Second 0: Dot lies at the start of the input as well
- NP → Det • Nominal, [1,2]
  - NP begins at position 1
  - Det has been successfully parsed
  - Nominal is expected next
- VP → V NP •, [0,3]
  - Successful discovery of a tree corresponding to a VP that spans the entire input

Earley Algorithm

- An Earley parser moves through the *n*+1 sets of states in a chart in order
- At each step, one of three operators is applied to each state depending on its status
  - Predictor
  - Scanner
  - Completer
- States can be added to the chart, but are never removed
- The algorithm never backtracks
- The presence of $S \rightarrow \alpha \bullet, [0,n]$ indicates a successful parse

Earley Operators: Predictor

Predictor

Creates new states

Applied to any state that has a non-terminal non-POS immediately to the right of its dot

New states are placed into the same chart entry as the generating state

They begin and end at the same point in the input where the generating state ends

S → • VP, [0,0]

VP → • Verb, [0,0]

VP → • Verb NP, [0,0]

VP → • Verb NP PP, [0,0]

VP → • Verb PP, [0,0]

VP → • VP PP, [0,0]

Earley Operators: Scanner

- Used when a state has a POS category to the right of the dot
- Examines input and (if relevant) adds a state predicting a word with a particular POS into the chart
- VP → • Verb NP, [0,0]
  - Since category following the dot is a part of speech (Verb)….
  - Verb → book •, [0,1]

Earley Operators: Completer

- Applied to a state when its dot has reached the right end of the rule
- Indicates that the parser has successfully discovered a grammatical category over some span of input
- Finds all previously created states that were searching for this grammatical category, and creates new states that are copies with their dots advanced past the grammatical category
- NP → Det Nominal •, [1,3]
    - What incomplete states end at position 1 and expect an NP?
    - VP → Verb • NP, [0,1]
    - VP → Verb • NP PP, [0,1]
    - So, add VP → Verb NP •, [0,3] and the new incomplete VP → Verb NP • PP, [0,3] to the chart

Earley Algorithm: Example

Chart	State	Rule	Start, End	Added By
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor

- Book that flight.

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
VP → Verb
VP → Verb NP

Earley Algorithm: Example

Book • that flight.

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
VP → Verb
VP → Verb NP

Chart	State	Rule	Start, End	Added By
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor

Earley Algorithm: Example

Book that • flight.

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
VP → Verb
VP → Verb NP

Chart	State	Rule	Start, End	Added By
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor
2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor

Earley Algorithm: Example

Book that flight. •

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
VP → Verb
VP → Verb NP

Chart	State	Rule	Start, End	Added By
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor
2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor
3	S14	Noun → flight •	2, 3	Scanner
3	S15	Nominal → Noun •	2, 3	Completer
3	S16	NP → Det Nominal •	1, 3	Completer
3	S17	VP → Verb NP •	0, 3	Completer
3	S18	S → VP •	0, 3	Completer

Which states participate in the final parse?

Chart	State	Rule	Start, End	Added By
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor
2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor
3	S14	Noun → flight •	2, 3	Scanner
3	S15	Nominal → Noun •	2, 3	Completer
3	S16	NP → Det Nominal •	1, 3	Completer
3	S17	VP → Verb NP •	0, 3	Completer
3	S18	S → VP •	0, 3	Completer

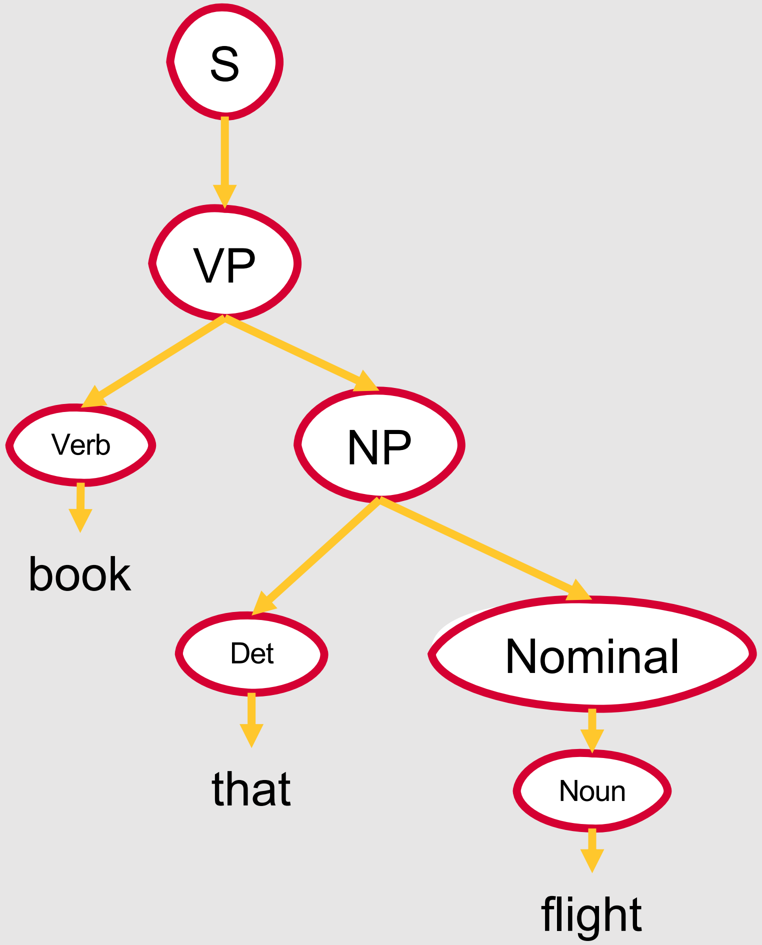
Which states participate in the final parse?

- We can retrieve parse trees by adding a field to store information about the completed states that generated constituents
- How to do this?
  - Have the Completer add a pointer to the previous state onto a list of constituent states for the new state
  - When an S is found in the final chart, just follow pointers backward

Chart	State	Rule	Start, End	Added By (Backward Pointer)
0	S0	$\gamma \rightarrow$ • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor
2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor
3	S14	Noun → flight •	2, 3	Scanner
3	S15	Nominal → Noun •	2, 3	Completer (S14)
3	S16	NP → Det Nominal •	1, 3	Completer (S11, S15)
3	S17	VP → Verb NP •	0, 3	Completer (S6, S16)
3	S18	S → VP •	0, 3	Completer (S17)

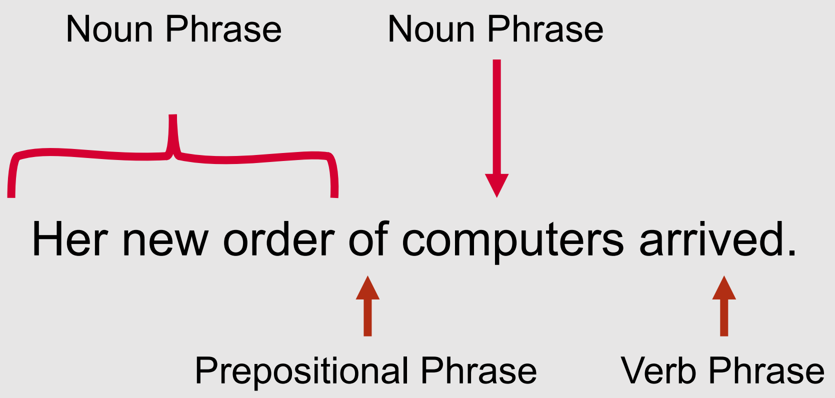
Successful Final Parse

Chart	State	Rule	Start, End	Added By (Backward Pointer)
0	S0	$\gamma \rightarrow \bullet$ S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor
2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor
3	S14	Noun → flight •	2, 3	Scanner
3	S15	Nominal → Noun •	2, 3	Completer (S14)
3	S16	NP → Det Nominal •	1, 3	Completer (S11, S15)
3	S17	VP → Verb NP •	0, 3	Completer (S6, S16)
3	S18	S → VP •	0, 3	Completer (S17)



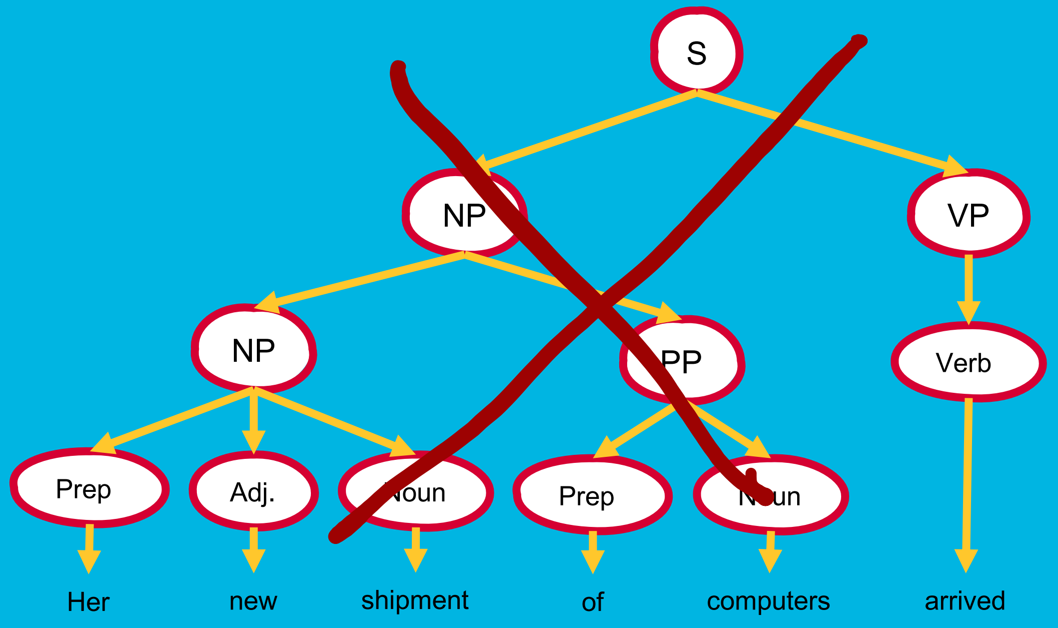
What if we don’t need a full parse tree?

- Full parse trees can be complex and time-consuming to build
- Many NLP tasks don’t require full hierarchical parses



Easier solution?

- **Partial parsing**, or **shallow parsing**
- How to generate a partial parse?
    - **Chunking**



[Her new shipment]$_{NP}$ [of]$_{PP}$ [computers]$_{NP}$ [arrived]$_{VP}$

Segmentation: Identify the non-overlapping, fundamental phrases

[Her new order] [of] [computers] [arrived]

Labeling: Assign labels to those phrases

[Her new order]$_{NP}$ [of]$_{PP}$ [computers]$_{NP}$ [arrived]$_{VP}$

Chunking: Fundamental Tasks

What is, and is not, a chunk?

- Non-recursive span of text
- When chunking phrases that would otherwise be parsed recursively:
  - Keep head word
  - Keep all material belonging to constituent that occurs before the head word

[Her new shipment of computers]$_{NP}$ [arrived]$_{VP}$

[Her new shipment]$_{NP}$ [of]$_{PP}$ [computers]$_{NP}$ [arrived]$_{VP}$

How do we segment text into spans?

- **IOB tagging**
  - **I:** Tokens **inside** a span
  - **O:** Tokens **outside** any span
  - **B:** Tokens **beginning** a span

Task: IOB Tagging (All Constituent Types)

B_NP I_NP B_NP

Her new order of computers arrived.

I_NP B_PP B_VP

Task: IOB Tagging (Noun Phrases)

B_NP I_NP B_NP

Her new order of computers arrived.

I_NP O O

How do we evaluate chunking systems?

- Standard text classification metrics, comparing predictions with a gold standard
  - Precision
  - Recall
  - F-measure

This Week's Topics

Tuesday

- Context-Free Grammars
- Syntactic Parsing
- CKY Algorithm

Thursday

- Earley Algorithm
- Probabilistic CKY
- Lexicalized Grammars

How can we resolve some of the parsing ambiguities we've observed?

- **Probabilistic Context-Free Grammars:** Can be used to determine which parse out of multiple valid parses should be selected, based on how likely the parse tree is to occur in a large corpus
- Same core components as regular CFGs:
  - A set of non-terminals, N
  - A set of terminal symbols, Σ
  - A set of rules or productions, R
  - A designated start symbol, S
- Each rule in R is of the form A → β, where A is a non-terminal and β is a string of symbols from the set $\Sigma \cup N$

How do PCFGs differ from CFGs?

- R is augmented with a probability, [p], learned from a corpus
- The sum of all probabilities for a given non-terminal is 1.0
- For example, if the following three expansions for S were possible, they might have the probabilities:
  - S $\rightarrow$ NP VP [0.80]
  - S $\rightarrow$ Aux NP VP [0.15]
  - S $\rightarrow$ VP [0.05]

Probabilistic Context-Free Grammars

- The probability of sentence S having a parse tree T is the product of the individual probabilities associated with its constituent rules
  - $P(T,S) = \prod_{i=1}^{n} P(\beta_i | A_i)$
- To disambiguate between multiple valid parses, we find the parse tree T that results in the highest probability for the sentence S
  - $\hat{T}(S) = \underset{T \ s.t. \ S=\text{yield}(T)}{\text{argmax}} P(T)$
- We can compute the probabilities for our parse trees by extending the parsing algorithms we already have

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

Still assume grammar is in Chomsky normal form!

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)				
1		N (0.01)			
2					
3					
4					

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)				
1		N (0.01)			
2			V (0.05)		
3				Det (0.40)	
4					

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)				
1		N (0.01)			
2			V (0.05)		
3				Det (0.40)	
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)			
1		N (0.01)			
2			V (0.05)		
3				Det (0.40)	
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)			
1		N (0.01)	🙁		
2			V (0.05)		
3				Det (0.40)	
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)			
1		N (0.01)	☹		
2			V (0.05)	☹	
3				Det (0.40)	
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)			
1		N (0.01)	☹		
2			V (0.05)	☹	
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

> The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)			
1		N (0.01)	☹		
2			V (0.05)	☹	
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)			
1		N (0.01)	☹️		
2			V (0.05)	☹️	
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

> The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	🙁		
1		N (0.01)	🙁		
2			V (0.05)	🙁	
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹		
1		N (0.01)	☹		
2			V (0.05)	☹	
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹		
1		N (0.01)	☹		
2			V (0.05)	☹	
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹		
1		N (0.01)	☹	☹	
2			V (0.05)	☹	
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹		
1		N (0.01)	☹	☹	
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.02 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

> The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹️		
1		N (0.01)	☹️	☹️	
2			V (0.05)	☹️	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹		
1		N (0.01)	☹	☹	
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹️		
1		N (0.01)	☹️	☹️	
2			V (0.05)	☹️	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹		
1		N (0.01)	☹	☹	
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹		
1		N (0.01)	☹	☹	
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

> The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹	☹	
1		N (0.01)	☹	☹	
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹️	☹️	
1		N (0.01)	☹️	☹️	
2			V (0.05)	☹️	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	🙁	🙁	
1		N (0.01)	🙁	🙁	
2			V (0.05)	🙁	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹	☹	
1		N (0.01)	☹	☹	
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹️	☹️	
1		N (0.01)	☹️	☹️	☹️
2			V (0.05)	☹️	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹	☹	
1		N (0.01)	☹	☹	☹
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹	☹	S (0.8 * 0.0012 * 0.000024 = $2.304*10^{-8}$)
1		N (0.01)	☹	☹	☹
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	☹	☹	S (0.8 * 0.0012 * 0.000024 = $2.304*10^{-8}$)
1		N (0.01)	☹	☹	☹
2			V (0.05)	☹	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	🙁	🙁	S (0.8 * 0.0012 * 0.000024 = $2.304*10^{-8}$)
1		N (0.01)	🙁	🙁	🙁
2			V (0.05)	🙁	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Case Example: Probabilistic CKY

The price includes a computer

Production Rule	Probability
S → NP VP	0.80
NP → Det N	0.30
VP → V NP	0.20
V → includes	0.05
Det → the	0.40
Det → a	0.40
N → price	0.01
N → computer	0.02

	0 - The	1 - price	2 - includes	3 - a	4 - computer
0	Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0012)	🙁	🙁	S (0.8 * 0.0012 * 0.000024 = $2.304*10^{-8}$)
1		N (0.01)	🙁	🙁	🙁
2			V (0.05)	🙁	VP (0.2 * 0.05 * 0.0024 = 0.000024)
3				Det (0.40)	NP (0.3 * 0.4 * 0.01 = 0.0024)
4					N (0.02)

Where did these probabilities come from?

- Often, a corpus
  - $P(\alpha \rightarrow \beta|\alpha) = \frac{Count(\alpha\rightarrow\beta)}{\sum_{\gamma} Count(\alpha\rightarrow\gamma)} = \frac{Count(\alpha\rightarrow\beta)}{Count(\alpha)}$
- Or, if we don't have a labeled corpus, we can apply a generalization of the forward-backward algorithm called the **inside-out algorithm**

Challenges Associated with PCFGs

- PCFGs solve many issues associated with resolving ambiguities, but they still have:
  - **Poor independence assumptions**, which may make it difficult to model important **structural dependencies** in the parse tree
  - **Lack of lexical conditioning**, which may allow **lexical dependency issues** (e.g., those dealing with preposition attachment or other syntactic properties) to arise
- More sophisticated techniques are needed, such as:
  - Adding extra constraints to rules by splitting them based on their parents or their syntactic positions
  - Using slightly different grammatical paradigms, such as **probabilistic lexicalized CFGs**

This Week's Topics

Tuesday

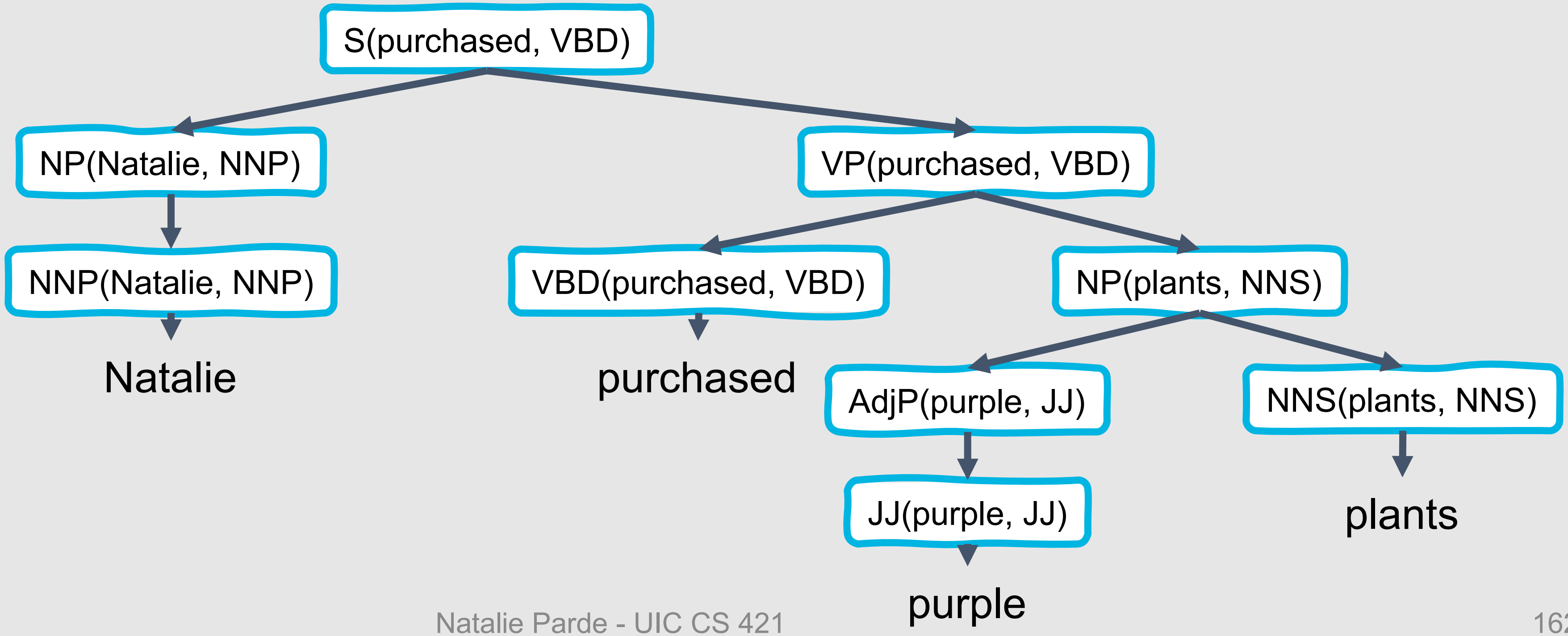
- Context-Free Grammars
- Syntactic Parsing
- CKY Algorithm

Thursday

- Earley Algorithm
- Probabilistic CKY
- Lexicalized Grammars

Lexicalized Parsers

- Allow lexicalized rules
  - Non-terminals specify lexical heads and associated POS tags
  - NP(plants, NNS) → AdjP(purple, JJ) NNS(plants, NNS)

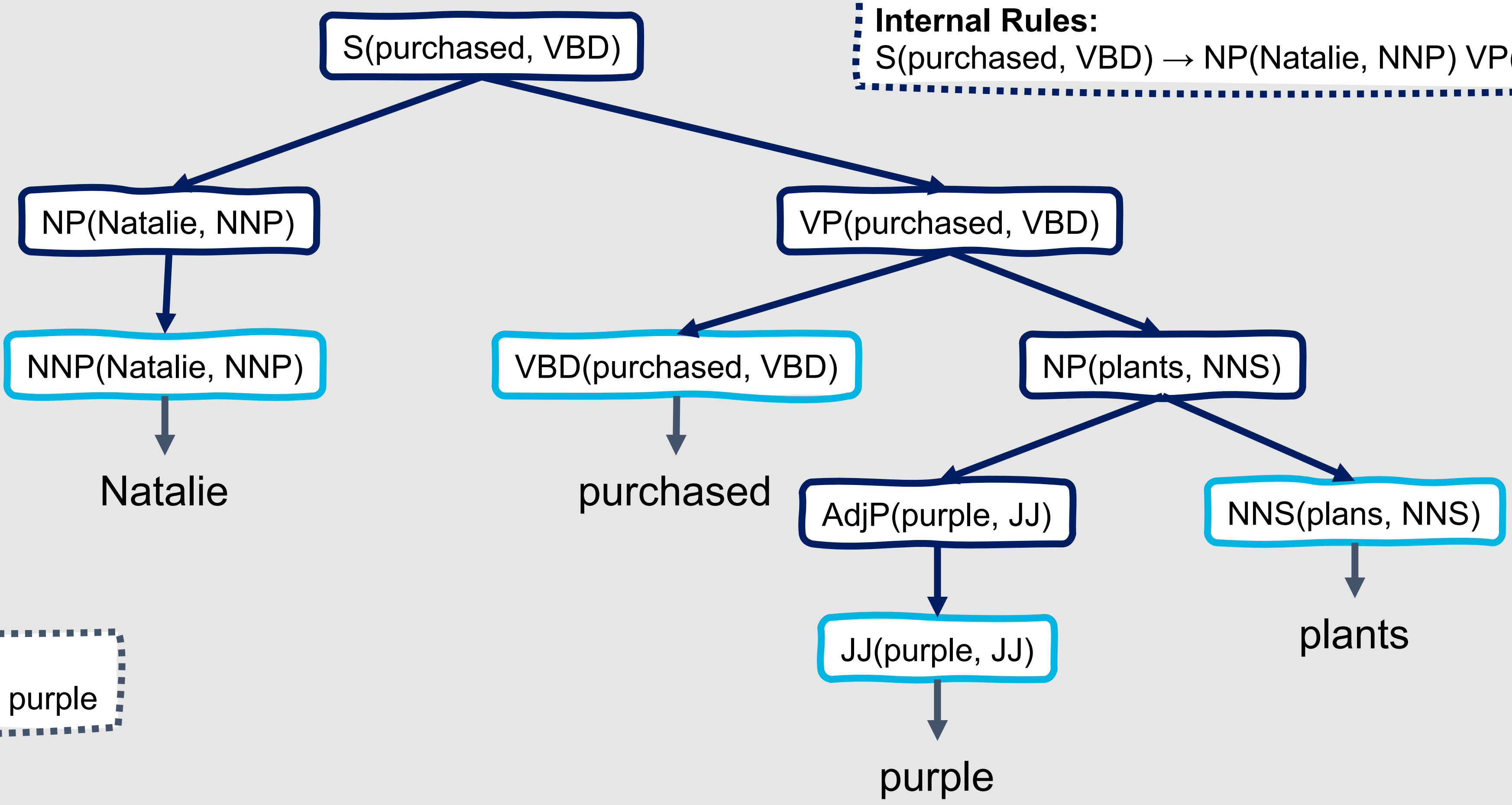


Lexicalized Grammars

- Intuitively, much like having many copies of the same production rule
  - NP(plants, NNS) → AdjP(purple, JJ) NNS(plants, NNS)
  - NP(plants, NNS) → AdjP(green, JJ) NNS(plants, NNS)
  - NP(computers, NNS) → AdjP(purple, JJ) NNS(computers, NNS)
- Two types of rules:
  - **Lexical Rules:** Generate a terminal word
    - Deterministic
  - **Internal Rules:** Generate a non-terminal constituent
    - Require estimated probabilities

Lexical vs. Internal Rules

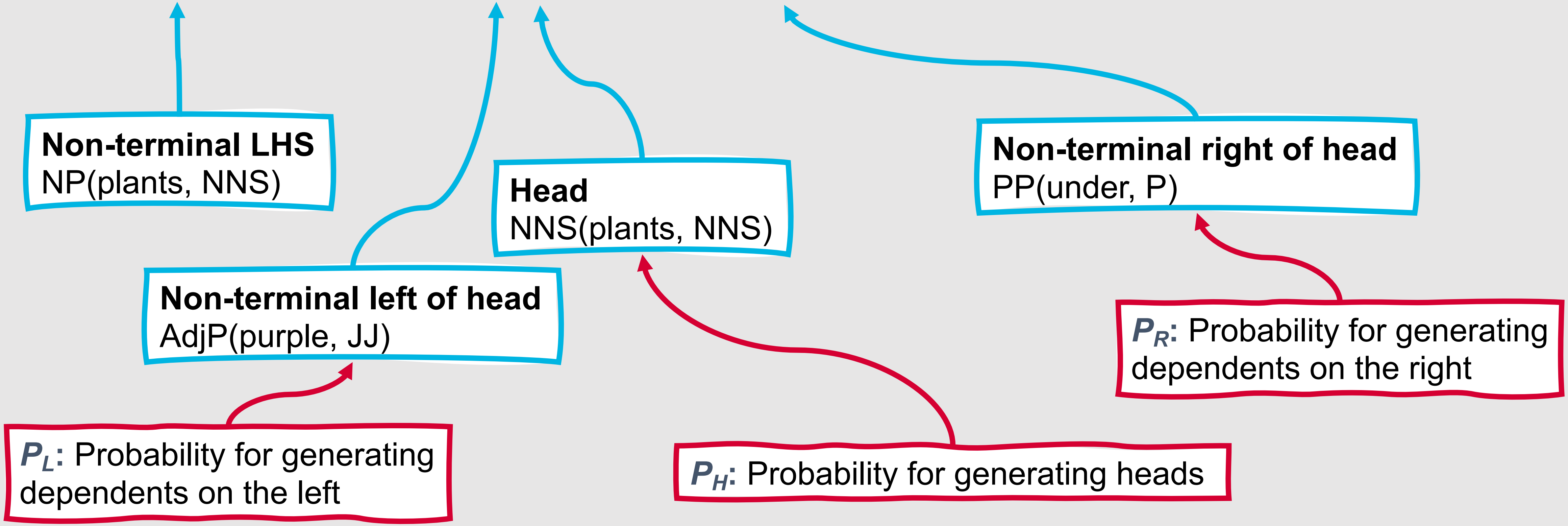
Internal Rules:
S(purchased, VBD) → NP(Natalie, NNP) VP(purchased, VBD)



Lexical Rules:
JJ(purple, JJ) → purple

The Collins Parser

- Consider the following generic production rule:
  - $LHS \rightarrow L_n\ L_{n-1} \dots L_1\ H\ R_1 \dots R_{n-1}\ R_n$



The Collins Parser

- Goal: Use P_H, P_L, and P_R to estimate the overall probability for the production rule
- Method:
  - Surround the righthand side of the rule with STOP non-terminals
    - NP(plants, NNS) → STOP AdjP(purple, JJ) NNS(plants, NNS) PP(under, IN) STOP
  - Compute the individual P_H, P_L, and P_R values for the head and the non-terminals to its left and right (including STOP non-terminals)
  - Multiply these together

Grab the **purple plants under the bookcase**.

The Collins Parser

- Consider the following generic production rule:
  - $LHS \rightarrow L_n\ L_{n-1} \ldots L_1\ H\ R_1 \ldots R_{n-1}\ R_n$

Non-terminal LHS
NP(plants, NNS)

Non-terminal left of head
AdjP(purple, JJ)

Head
NNS(plants, NNS)

Non-terminal right of head
PP(under, IN)

P_L: Probability for generating dependents on the left

P_H: Probability for generating heads

P_R: Probability for generating dependents on the right

Grab the **purple plants under the bookcase**.

NP(plants, NNS) → STOP AdjP(purple, JJ) NNS(plants, NNS) PP(under, IN) STOP

The Collins Parser

- Consider the following generic production rule:
  - $LHS \rightarrow L_n \; L_{n-1} \ldots L_1 \; H \; R_1 \ldots R_{n-1} \; R_n$

Non-terminal LHS
NP(plants, NNS)

Non-terminal left of head
AdjP(purple, JJ)

Head
NNS(plants, NNS)

Non-terminal right of head
PP(under, IN)

P_L: Probability for generating dependents on the left

P_H: Probability for generating heads

P_R: Probability for generating dependents on the right

$P_H(H|LHS)$ = P(NNS(plants, NNS) | NP(plants, NNS))

Grab the **purple plants under the bookcase**.

NP(plants, NNS) → STOP AdjP(purple, JJ) **NNS(plants, NNS)** PP(under, IN) STOP

The Collins Parser

- Consider the following generic production rule:
  - $LHS \rightarrow L_n \; L_{n-1} \ldots L_1 \; H \; R_1 \ldots R_{n-1} \; R_n$

Non-terminal LHS
NP(plants, NNS)

Non-terminal left of head
AdjP(purple, JJ)

Head
NNS(plants, NNS)

Non-terminal right of head
PP(under, IN)

P_L: Probability for generating dependents on the left

P_H: Probability for generating heads

P_R: Probability for generating dependents on the right

Grab the **purple plants under the bookcase**.

NP(plants, NNS) → **STOP AdjP(purple, JJ)** NNS(plants, NNS) PP(under, IN) STOP

P_H(H|LHS) = P(NNS(plants, NNS) | NP(plants, NNS))

P_L(STOP|LHS H) = P(STOP | NP(plants, NNS) NNS(plants, NNS))

P_L(L_1|LHS H) = P(AdjP(purple, JJ) | NP(plants, NNS) NNS(plants, NNS))

The Collins Parser

- Consider the following generic production rule:
  - $LHS \rightarrow L_n\ L_{n-1} \ldots L_1\ H\ R_1 \ldots R_{n-1}\ R_n$

Non-terminal LHS
NP(plants, NNS)

Non-terminal left of head
AdjP(purple, JJ)

Head
NNS(plants, NNS)

Non-terminal right of head
PP(under, IN)

P_L: Probability for generating dependents on the left

P_H: Probability for generating heads

P_R: Probability for generating dependents on the right

Grab the **purple plants under the bookcase**.

NP(plants, NNS) → STOP AdjP(purple, JJ) NNS(plants, NNS) **PP(under, IN) STOP**

P_H(H|LHS) = P(NNS(plants, NNS) | NP(plants, NNS))

P_L(STOP|LHS H) = P(STOP | NP(plants, NNS) NNS(plants, NNS))

P_L(L_1|LHS H) = P(AdjP(purple, JJ) | NP(plants, NNS) NNS(plants, NNS))

P_R(R_1|LHS H) = P(PP(under, IN) | NP(plants, NNS) NNS(plants, NNS))

P_R(STOP|LHS H) = P(STOP | NP(plants, NNS) NNS(plants, NNS))

The Collins Parser

- Consider the following generic production rule:
  - $LHS \rightarrow L_n \; L_{n-1} \ldots L_1 \; H \; R_1 \ldots R_{n-1} \; R_n$

Non-terminal LHS
NP(plants, NNS)

Non-terminal left of head
AdjP(purple, JJ)

Head
NNS(plants, NNS)

Non-terminal right of head
PP(under, IN)

P_L: Probability for generating dependents on the left

P_H: Probability for generating heads

P_R: Probability for generating dependents on the right

Grab the **purple plants under the bookcase**.

NP(facemasks, NNS) → STOP AdjP(purple, JJ) NNS(plants, NNS) **PP(under, IN) STOP**

= P_H(H|LHS) * P_L(STOP|LHS H) * P_L(L_1|LHS H) * P_R(R_1|LHS H) * P_R(STOP|LHS H)

P_H(H|LHS) = P(NNS(plants, NNS) | NP(plants, NNS))

P_L(STOP|LHS H) = P(STOP | NP(plants, NNS) NNS(plants, NNS))

P_L(L_1|LHS H) = P(AdjP(purple, JJ) | NP(plants, NNS) NNS(plants, NNS))

P_R(R_1|LHS H) = P(PP(under, IN) | NP(plants, NNS) NNS(plants, NNS))

P_R(STOP|LHS H) = P(STOP | NP(plants, NNS) NNS(plants, NNS))

Estimate the individual probabilities using maximum likelihood estimates.

$P_R(R_1|\text{LHS H}) = P(\text{PP(under, IN)} \mid \text{NP(plants, NNS) NNS(plants, NNS)})$

$$\frac{\text{Count}(\text{NP}(\text{plants}, \text{NNS})\ \textit{with}\ \text{PP}(\text{under}, \text{IN})\ \textit{as a child to the right})}{\text{Count}(\text{NP}(\text{plants}, \text{NNS}))}$$

Combinatory Categorial Grammars (CCGs)

- *Heavily* lexicalized approach that groups words into categories and defines ways that those categories may be combined
- Three major parts:
  - Categories
  - Lexicon
  - Rules

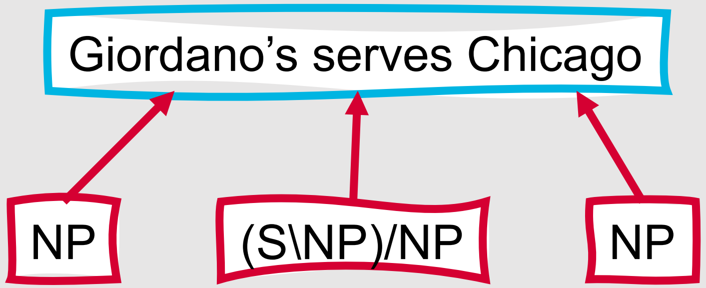
CCG Categories

- **Atomic elements**
  - $\mathcal{A} \subseteq \mathcal{C}$, where $\mathcal{A}$ is a set of atomic elements, and $\mathcal{C}$ is the set of categories for the grammar
  - Simple noun phrases
- **Single-argument functions**
  - (X/Y), (X\Y) $\in \mathcal{C}$, if $X, Y \in \mathcal{C}$
    - (X/Y): Seeks a constituent of type Y to the right, and returns X
    - (X\Y): Seeks a constituent of type Y to the left, and returns X
  - Verb phrases, more complex noun phrases, etc.

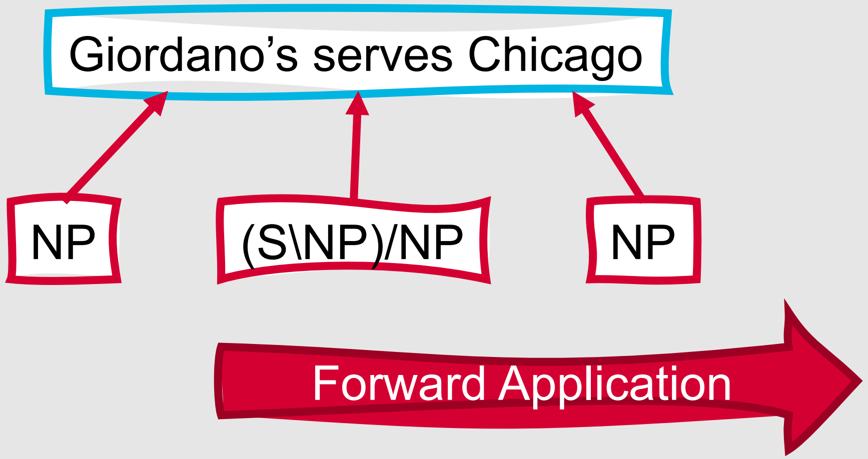
CCG Lexica and Rules

- CCG lexica assign CCG categories to words
  - Chicago: NP
    - **Atomic category**
  - cancel: (S\NP)/NP
    - **Functional category**
    - Seeks an NP to the right, returning (S\NP), which seeks an NP to the left, returning S
- CCG rules specify how functions and their arguments may be combined
  - **Forward function application:** Applies the function to its argument on the right, resulting in the specified category
    - $X/Y \; Y \Rightarrow X$
  - **Backward function application:** Applies the function to its argument on the left, resulting in the specified category
    - $Y \; X\backslash Y \Rightarrow X$
  - A coordination rule can also be applied
    - $X \; CONJ \; X \Rightarrow X$

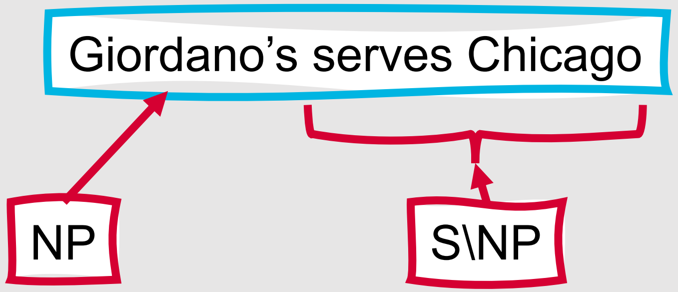
CCG Rules: Example



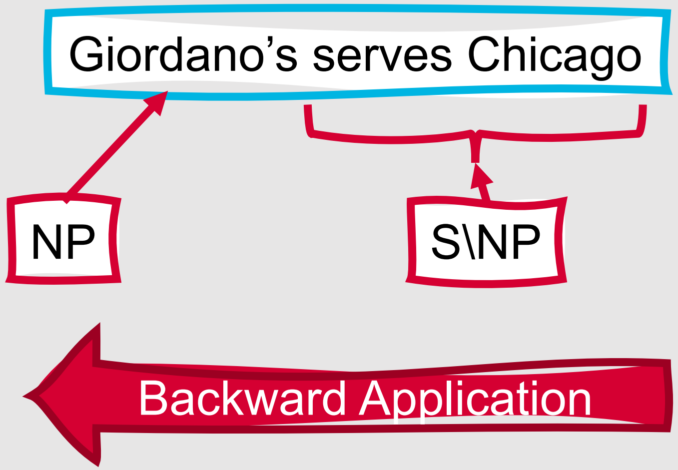
CCG Rules: Example



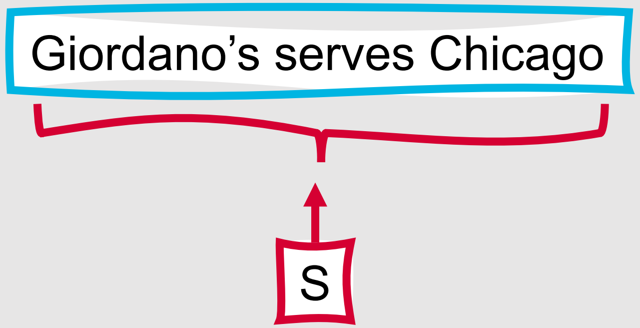
CCG Rules: Example



CCG Rules: Example



CCG Rules: Example



CCG Operations

- **CCG operations are forward and backward compositional**
  - X/Y Y/Z ⇒ X/Z
  - Y\Z X\Y ⇒ X\Z
- **Type raising**
  - Converts atomic categories to functional categories, or simple functional categories to more complex functional categories
    - X ⇒ T/(T\X), where T can be any existing atomic or functional category
    - X ⇒ T\(T/X)
  - Facilitates the creation of intermediate elements that do not directly map to traditional constituents in the language
- Type raising and function composition can be employed together to parse **long-range dependencies**

CCG Parsing Frameworks

Probabilistic CKY

- Works okay, but needs to be adapted a bit due to the large number of categories available for each word (otherwise, lots of unnecessary constituents would be added to the table)
- The solution: **Supertagging**
  - Trained using CCG treebanks (e.g., CCGBank)
  - Predict allowable category assignments (supertags) for each word in a lexicon, given an input context

A* Algorithm

- Heuristic search algorithm that finds the lowest-cost path to an end state, by exploring the lowest-cost partial solution at each iteration until a full solution is identified
- Search states = edges representing completed constituents
- Cost is based on the probability of the CCG derivation
- Results in fewer unnecessary constituents being explored than probabilistic CKY

Evaluating Parsers

- **PARSEVAL measures:** Seek to determine how close a predicted parse is to a gold standard parse for the same text, based on its individual constituents
  - Constituent is correct if it matches a constituent in the gold standard in terms of its:
    - Starting point
    - Ending point
    - Non-terminal symbol

Once constituent correctness is defined….

- We can apply the same metrics we use for other NLP problems!
  - Recall = $\frac{\#\text{ correct constituents in predicted parse}}{\#\text{ constituents in gold standard parse}}$
  - Precision = $\frac{\#\text{ correct constituents in predicted parse}}{\#\text{ constituents in predicted parse}}$

Summary: Statistical Constituency Parsing

The **Earley** algorithm is a top-down dynamic programming approach for syntactic parsing

We can select the best parse for a sentence using **probabilistic context-free grammars**

The **CKY algorithm** can be updated to incorporate these probabilities for use with PCFG parsing

An alternative parsing paradigm uses **lexicalized grammar frameworks**

We can evaluate parsers using standard NLP metrics applied based on the number of **correctly identified constituents** in a predicted parse