Introduction to Parsing, Context Free Grammar

(slides adapted and/or stolen outright from Chloe Kiddon, Andrew McCallum, Christopher Manning, and Julia Hockenmaier)

Syntax

- Refers to the study of the way words are arranged together, and the relationship between them.

- Prescriptive vs. Descriptive

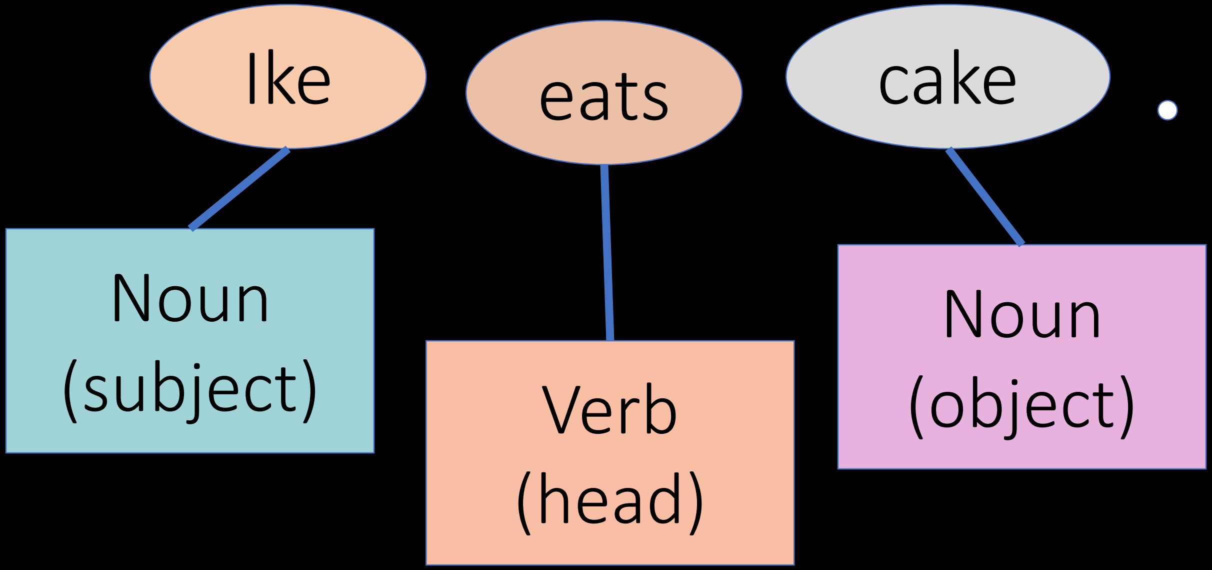
- **Goal of syntax is to model the knowledge of that people unconsciously have about the grammar of their native language**

- Parsing extracts the syntax from a sentence

Parsing applications

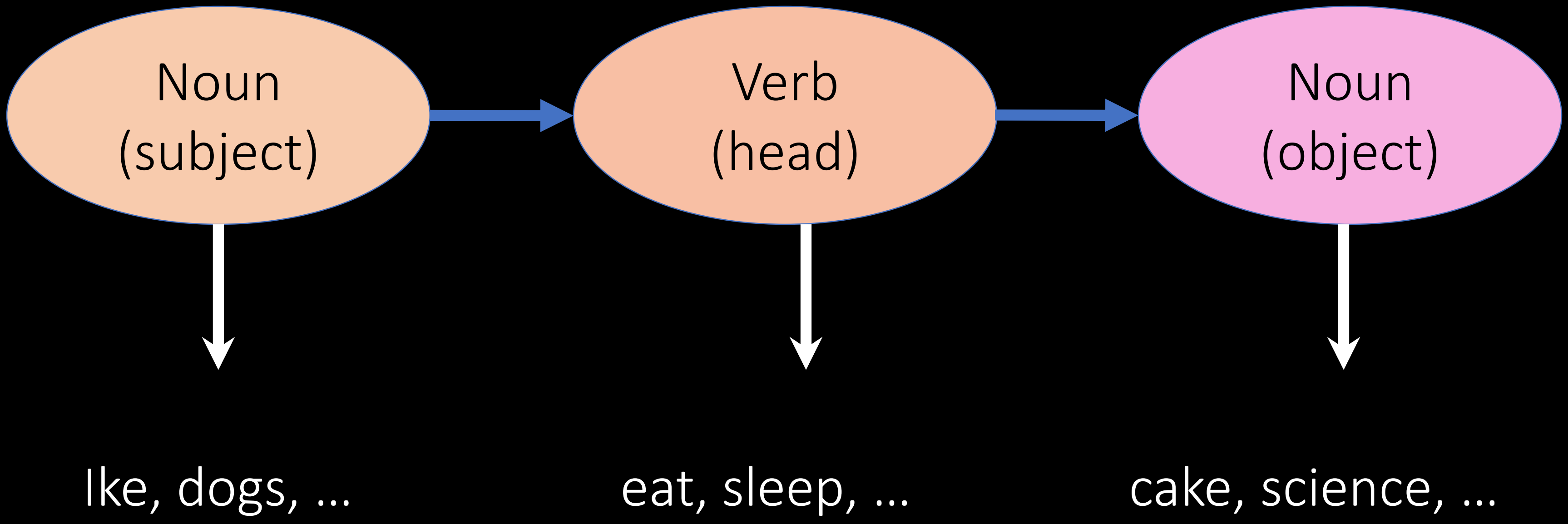
- High-precision Question-Answering systems
- Named Entity Recognition (NER) and information extraction
- Opinion extraction in product reviews
- Improved interaction during computer applications/games

Basic English sentence structure



Hockenmaier

Can we build an HMM?



Noun (subject)

Verb (head)

Noun (object)

Ike, dogs, …

eat, sleep, …

cake, science, …

Hockenmaier

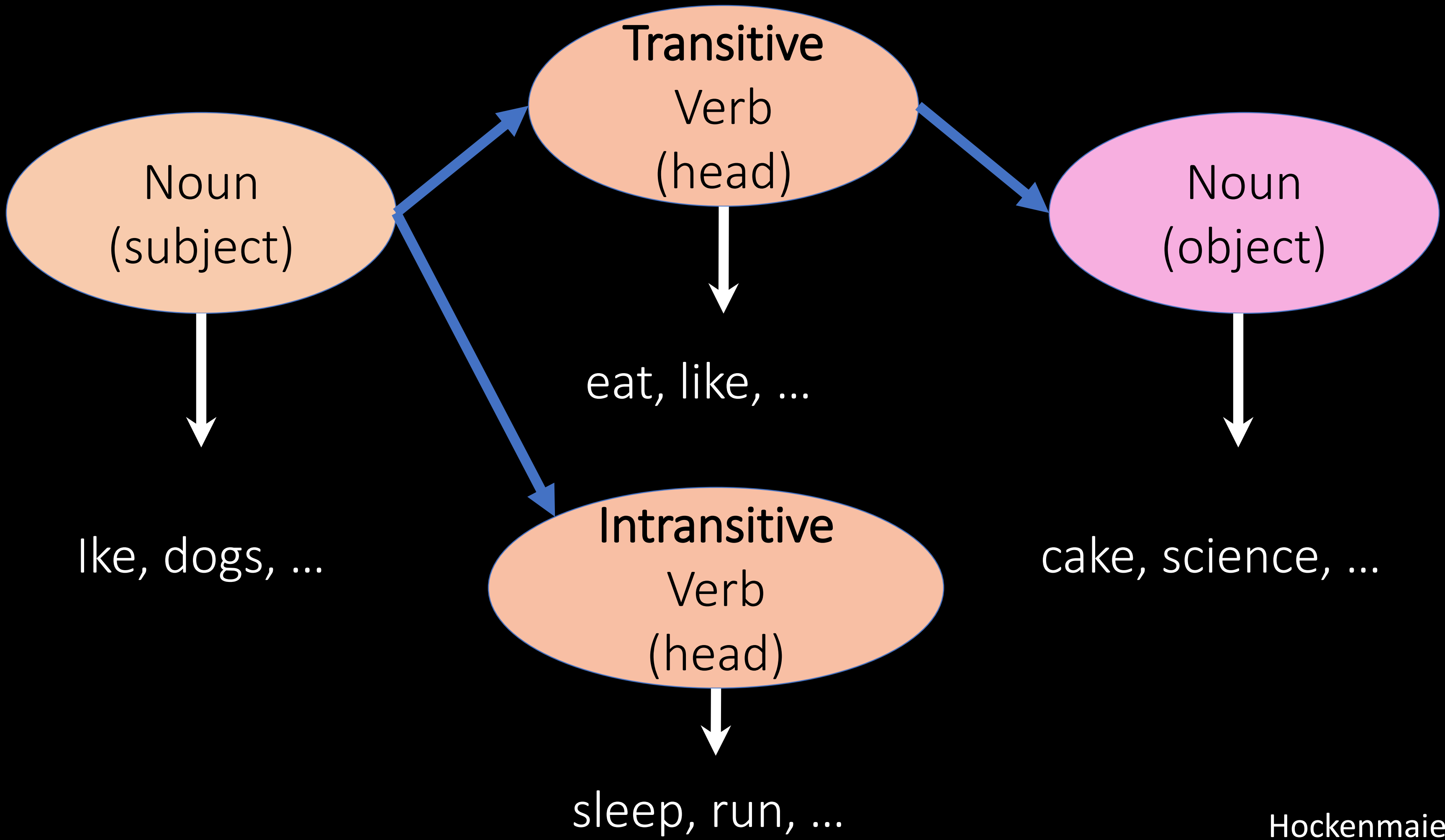
Words take arguments

I eat cake.	☺
I sleep cake.	☹
I give you cake.	☺
I give cake.	Hmm…
I eat you cake???	☹☹

- Subcategorization
    - **Intransitive verbs**: take only a subject
    - **Transitive verbs**: take a subject and an object
    - **Ditransitive verbs**: take a subject, object, and indirect object
- Selectional preferences

Hockenmaier

A better model



Noun (subject)

Ike, dogs, …

Transitive Verb (head)

eat, like, …

Intransitive Verb (head)

sleep, run, …

Noun (object)

cake, science, …

Hockenmaier

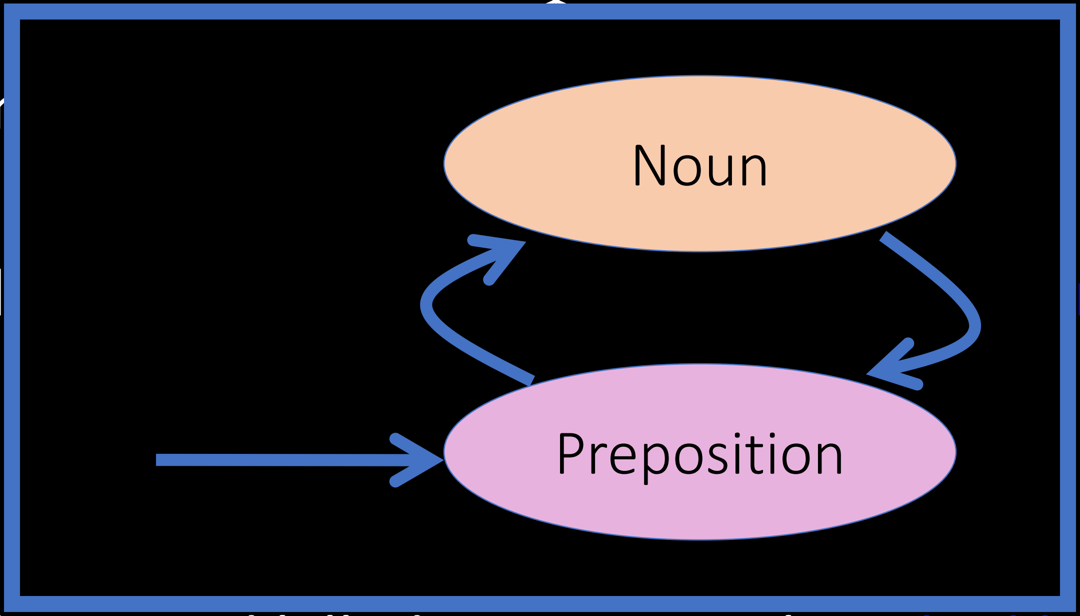
Language has recursive properties

Coronel Mustard killed Mrs. Peacock

*

Coror library

Coronel rary with



Coronel Mustard killed Mrs. Peacock in the library with the candlestick at midnight

HMMs can’t generate hierarchical structure

Coronel Mustard killed Mrs. Peacock in the library with the candlestick at midnight.

- Does Mustard have the candlestick?
- Or is the candlestick just sitting in the library?

- Memoryless
  - Can’t make long range decisions about attachments
- Need a better model

Words work in groups

- **Constituents** – words or groupings of words that function as single units
    - Noun phrases (NPs)
        - The computer science class
        - Peter, Paul, and Mary
        - PAC10 Schools, such as UW,
        - He
        - The reason I was late

Words work in groups

- **Constituents** – words or groupings of words that function as single units
  - Noun phrases (NPs)
    - The computer science class **listened …**
    - Peter, Paul, and Mary **sing …**
    - PAC10 Schools, such as UW, **dominate …**
    - He **juggled …**
    - The reason I was late **was …**
    - *the listened
    - *such sing
    - *late was

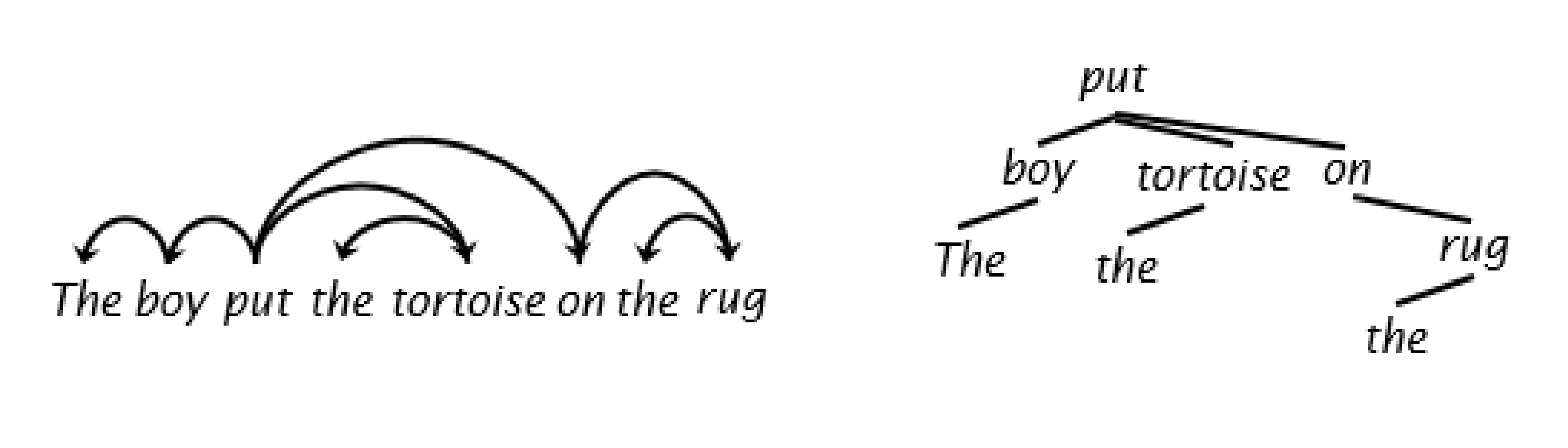
NPs can appear before a verb.

Two views of linguistic structure: 1. Constituency (phrase structure)

- Phrase structure organizes words into nested constituents.
- How do we know what is a constituent? (Not that linguists don’t argue about some cases.)
  - Distribution: a constituent behaves as a unit that can appear in different places:
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].
    - *John talked drugs to the children about
  - Substitution/expansion/pro-forms:
    - I sat [on the box/right on top of the box/there].
  - Coordination, regular internal structure, no intrusion, fragments, semantics, …

Two views of linguistic structure: 2. Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.



The boy put the tortoise on the rug

Attachment ambiguities

- Teacher Strikes Idle Kids
- Squad Helps Dog Bite Victim
- Complaints About NBA Referees Getting Ugly
- Soviet Virgin Lands Short of Goal Again
- Milk Drinkers are Turning to Powder

Attachment ambiguities

- The key parsing decision: How do we ‘attach’ various kinds of constituents – PPs, adverbial or participial phrases, coordinations, etc.
- Prepositional phrase attachment
    - *I saw the man with the telescope.*
- What does wit*h a telescope* modify?
    - The verb *saw*?
    - The noun *man*?
- Very hard problem. AI Complete.

Parsing

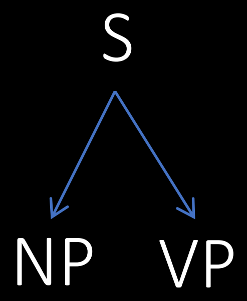
- We want to run a grammar backwards to find possible structures for a sentence

- Parsing can be viewed as a **search problem**

- Parsing is a **hidden data problem**

Context-free grammars (CFGs)

- Specifies a set of tree structures that capture constituency and ordering in language
  - A noun phrase can come before a verb phrase
    - S → NP VP



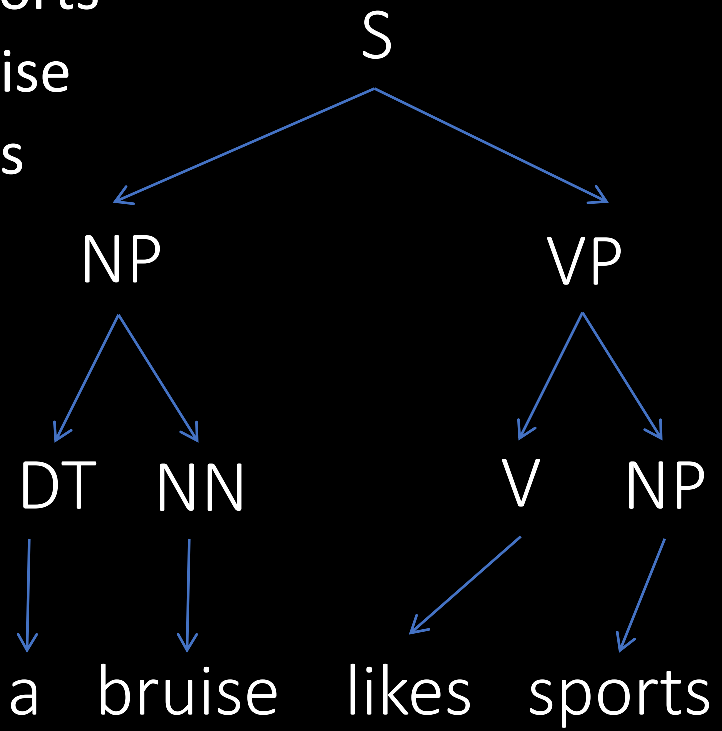
Phrase structure grammars = Context-free grammars

- G = (T, N, S, R)
  - T is the set of terminals (i.e. words)
  - N is the set of non-terminals
    - Usually separate the set P of preterminals (POS tags) from the rest of the non-terminals
    - S is the start symbol
    - R is the set of rules/productions of the form X → γ where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly empty)
- A grammar G generates a language L

A phrase structure grammar

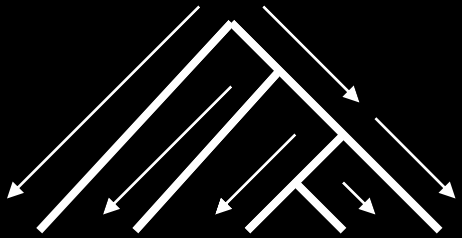
- By convention, S is the start symbol
  - S → NP VP
  - NP → DT NN
  - NP → NNS
  - VP → V NP
  - VP → V
  - …

NN → boy
NNS → sports
NN → bruise
V → sports
V → likes
DT → a

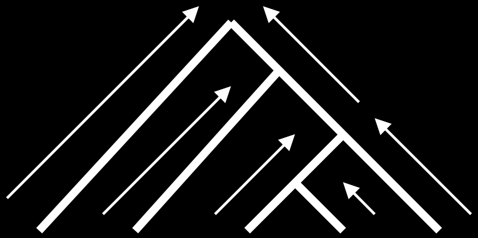


How to parse

- Top-down: Start at the top of the tree with an S node, and work your way down to the words.

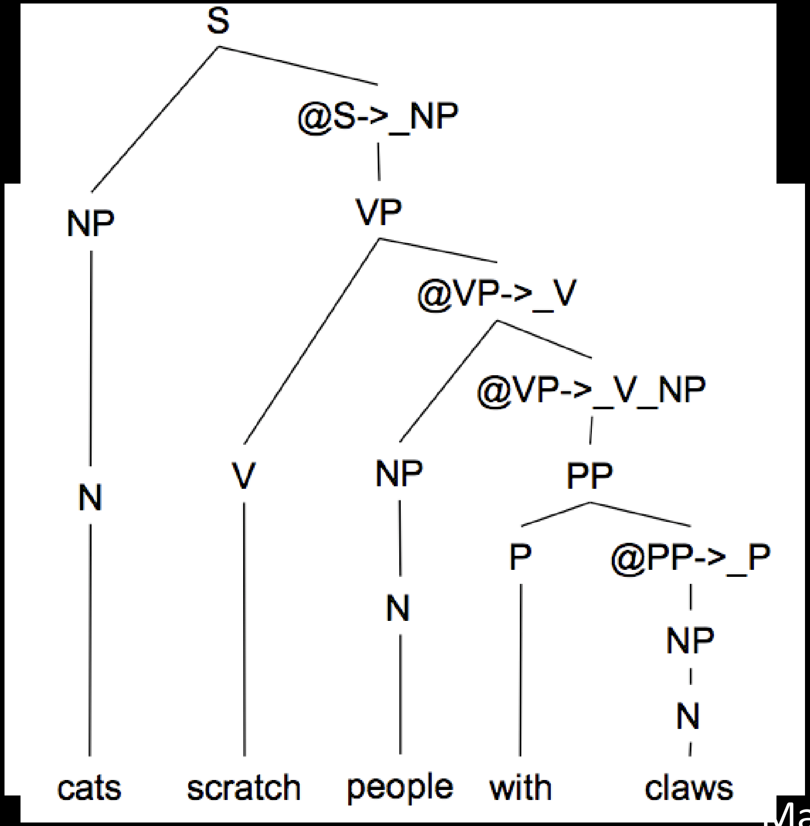


- Bottom-up: Look for small pieces that you know how to assemble, and work your way up to larger pieces.



Chomsky Normal Form

- All rules are of the form X → Y Z or X → w.
- *n*-ary rules introduce new nonterminals (n > 2)
  - VP → V NP PP
    becomes:
    VP → V @VP-V and
    @VP-V → NP PP



Manning

Top Down Parsing

LL Parsing, remember?

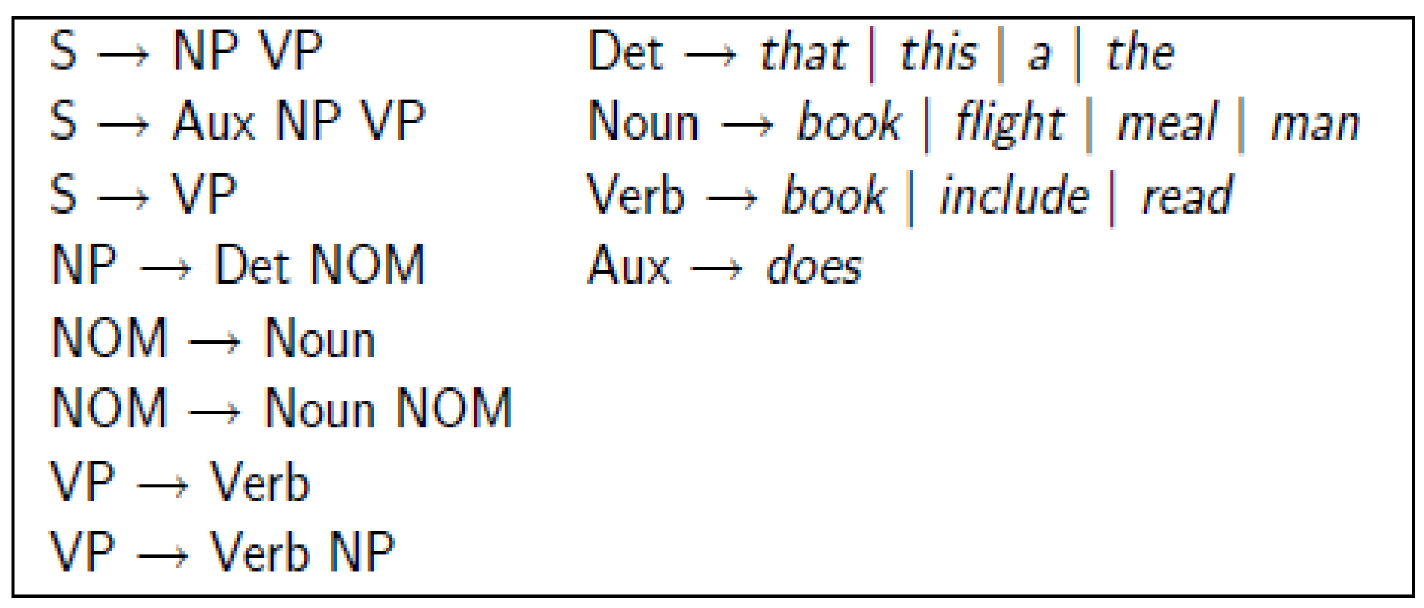
Top-down parsing is goal-directed.

- A top-down parser starts with a list of constituents to be built.
- It rewrites the goals in the goal list by matching one against the LHS of the grammar rules,
- and expanding it with the RHS,
- attempting to match the sentence to be derived.

If a goal can be rewritten in several ways, then there is a choice of which rule to apply (search problem)

Can use depth-first or breadth-first search, and goal ordering.

Top Down Parsing Example



S → NP VP
S → Aux NP VP
S → VP
NP → Det NOM
NOM → Noun
NOM → Noun NOM
VP → Verb
VP → Verb NP

Det → *that* | *this* | *a* | *the*
Noun → *book* | *flight* | *meal* | *man*
Verb → *book* | *include* | *read*
Aux → *does*

Sentence example: *Book that flight*

Top Down Parsing Problems

- Left recursive rules... e.g. NP ! NP PP... lead to infinite recursion
- Will do badly if there are many different rules for the same LHS. Consider if there are 600 rules for S, 599 of which start with NP, but one of which starts with a V, and the sentence starts with a V.
- Useless work: expands things that are possible top-down but not there (no bottom-up evidence for them).
- Top-down parsers do well if there is useful grammar-driven control: search is directed by the grammar.
- Top-down is hopeless for rewriting parts of speech (preterminals) with words (terminals). In practice that is always done bottom-up as lexical lookup.
- Repeated work: anywhere there is common substructure.

Bottom Up Parsing

LR Parsing, remember?

Bottom-up parsing is data-directed.

- The initial goal list of a bottom-up parser is the string to be parsed.
- If a sequence in the goal list matches the RHS of a rule, then this sequence may be replaced by the LHS of the rule.
- Parsing is finished when the goal list contains just the start symbol.

If the RHS of several rules match the goal list, then there is a choice of which rule to apply (search problem)

Can use depth-first or breadth-first search, and goal ordering.

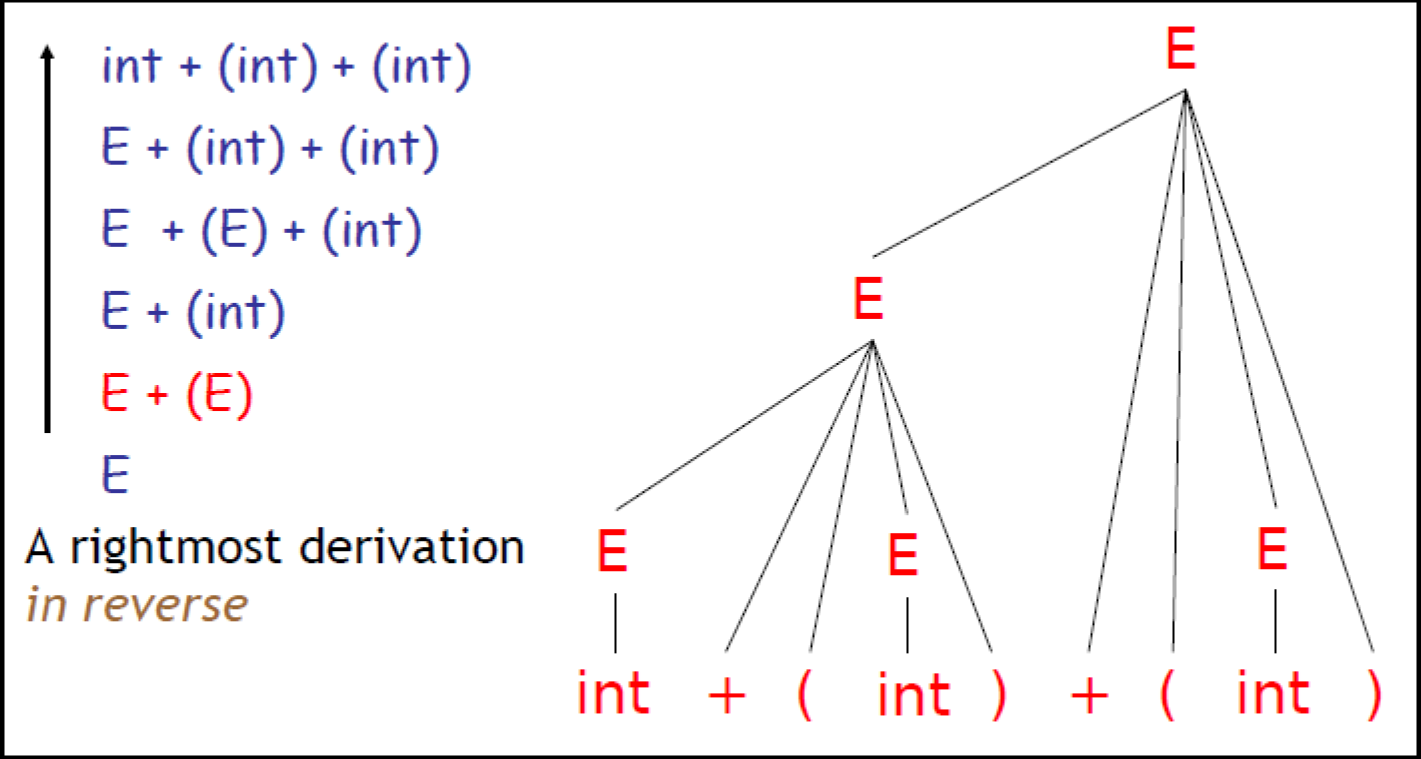
The standard presentation is as shift-reduce parsing.

The Idea of Bottom Up Parsing

- LR parsing **reduces** a string to the start symbol by *inverting* productions:

```
str <- input string of terminals
repeat
    • Identify β in str such that A -> β is a production
      (i.e., str = αβγ)
    • Replace β by A in str (i.e., str becomes αAγ)
until str = S
```

Example of Bottom Up Parse Detail



int + (int) + (int)

E + (int) + (int)

E + (E) + (int)

E + (int)

E + (E)

E

A rightmost derivation *in reverse*

Bottom Up Parsing: Shift

Shift: Move ▶ one place to the right

- –Shifts a terminal to the left string

E + (▶ int)

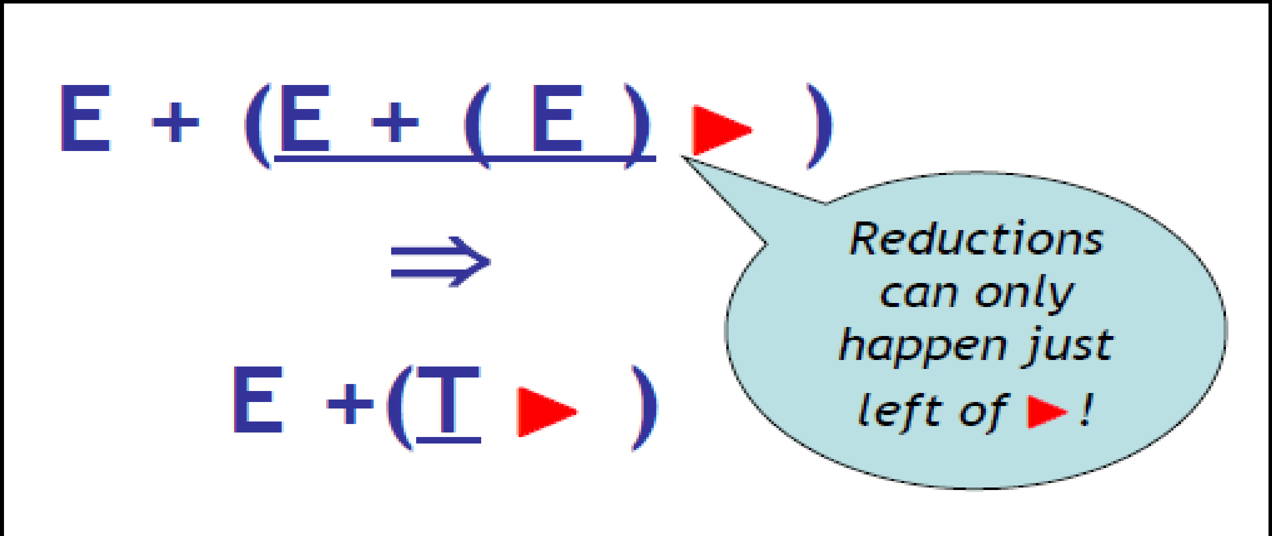
=>

E + (int ▶)

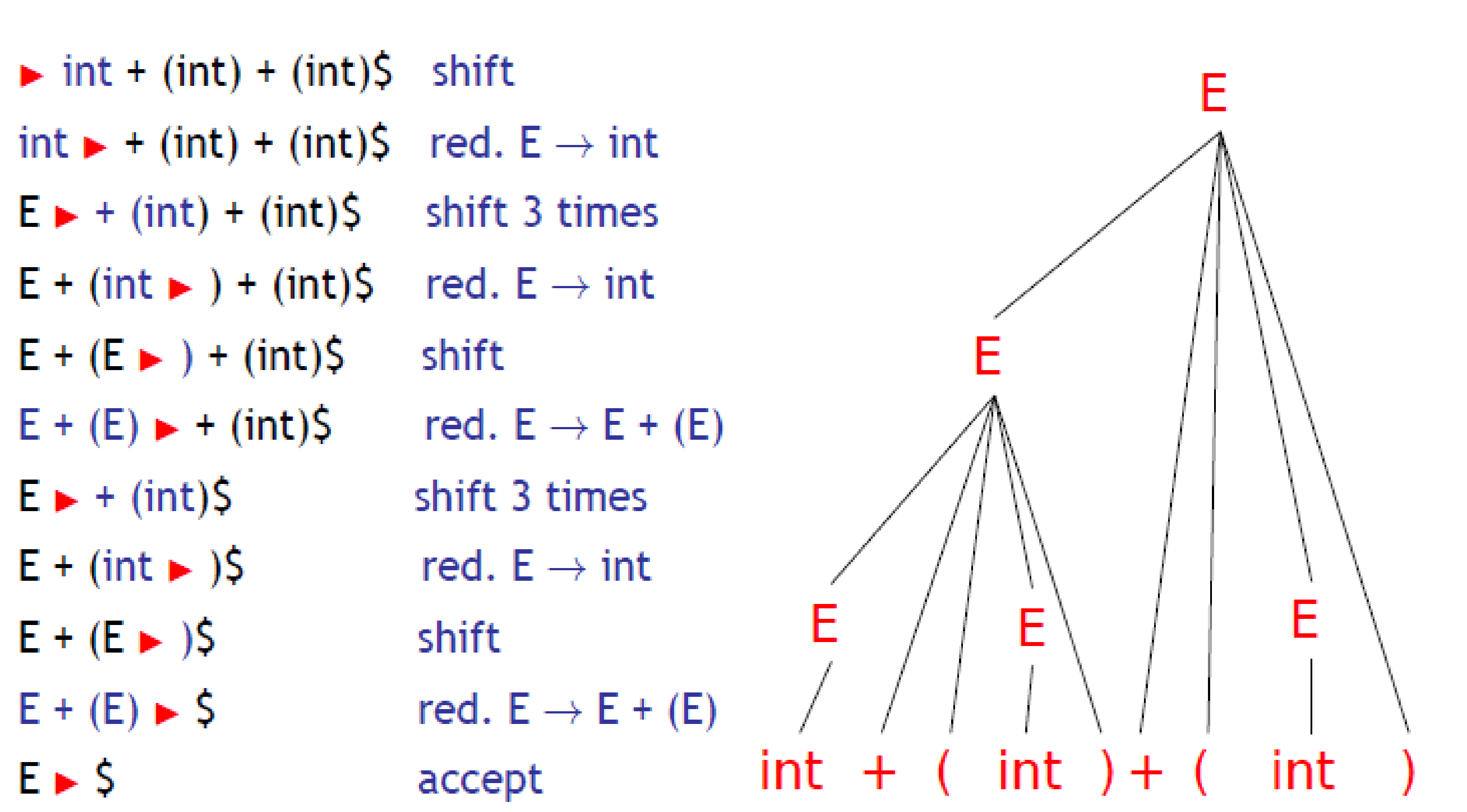
Bottom Up Parsing: Reduce

Reduce: Apply an inverse production at the right end of the left string

- –If **T -> E + (E)** is a production, then

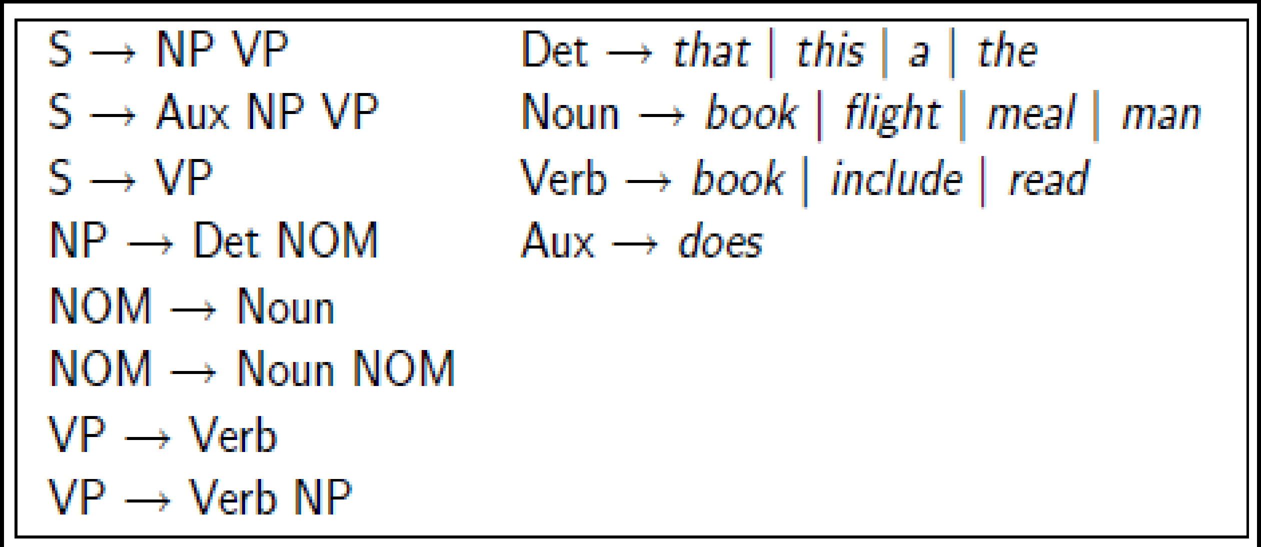


Shift Reduce Example



▸ int + (int) + (int)$	shift
int ▸ + (int) + (int)$	red. E → int
E ▸ + (int) + (int)$	shift 3 times
E + (int ▸) + (int)$	red. E → int
E + (E ▸) + (int)$	shift
E + (E) ▸ + (int)$	red. E → E + (E)
E ▸ + (int)$	shift 3 times
E + (int ▸)$	red. E → int
E + (E ▸)$	shift
E + (E) ▸ $	red. E → E + (E)
E ▸ $	accept

Bottom Up Parsing Example



S → NP VP
S → Aux NP VP
S → VP
NP → Det NOM
NOM → Noun
NOM → Noun NOM
VP → Verb
VP → Verb NP

Det → *that* | *this* | *a* | *the*
Noun → *book* | *flight* | *meal* | *man*
Verb → *book* | *include* | *read*
Aux → *does*

Sentence example: *Book that flight*

Huge area of research

- Coarse-to-fine parsing
  - Parse with a simpler grammar
  - Refine with a more complex one
- Dependency parsing
  - A sentence is parsed by relating each word to other words in the sentence which depend on it.
- Discriminative parsing
  - Given training examples, learn a function that classifies a sentence with its parse tree
- and more!

The good news!

- Part of speech taggers and sentence parsers are freely available!

- So why did we sit through this lecture?
  - Maybe you’ll be interested in this area
  - Useful ideas to be applied elsewhere
    - Write a parser to parse web tables
    - PCFGs for information extraction
  - Like to know how things work

It’s over!

- Thanks!