Foundations of Natural Language Processing
Lecture 13
Heads, Dependency parsing

Shay Cohen
(slides from Henry Thompson, Alex Lascarides,
Henry Thompson, Nathan Schneider and Sharon Goldwater)

3 March 2020

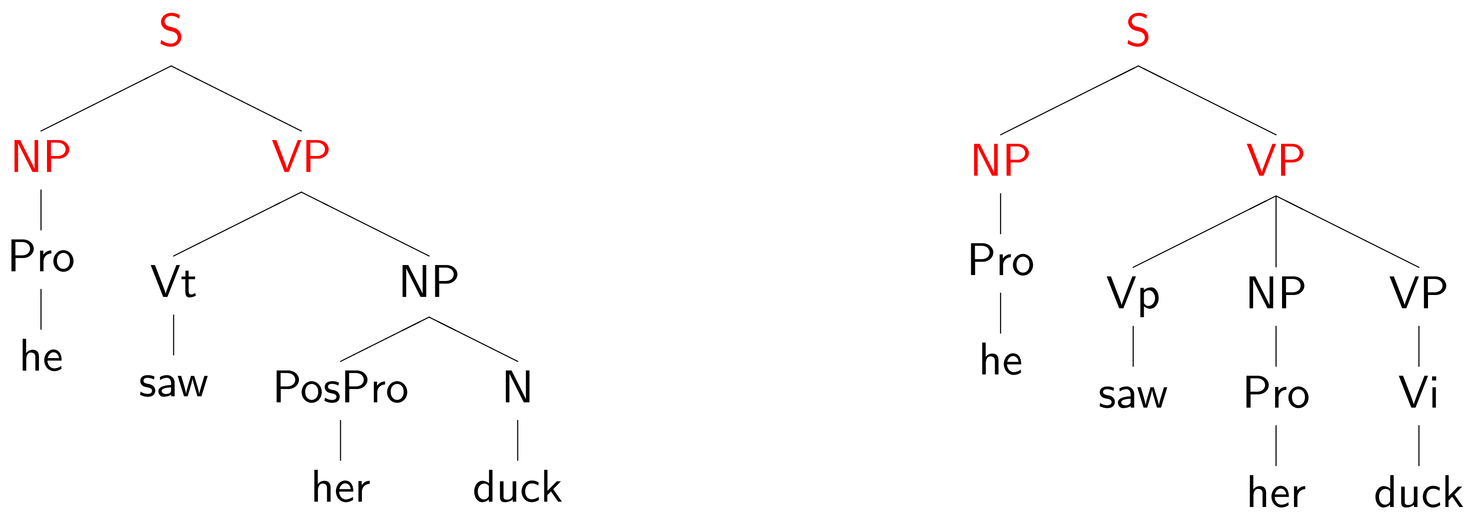
Today we will. . .

- Provide metrics for evaluating a parser
- Return to the problem of PCFGs
- Suggest a fix
- This fix leads to an approach without constituent structure!

Dependency parsing

Evaluating parse accuracy

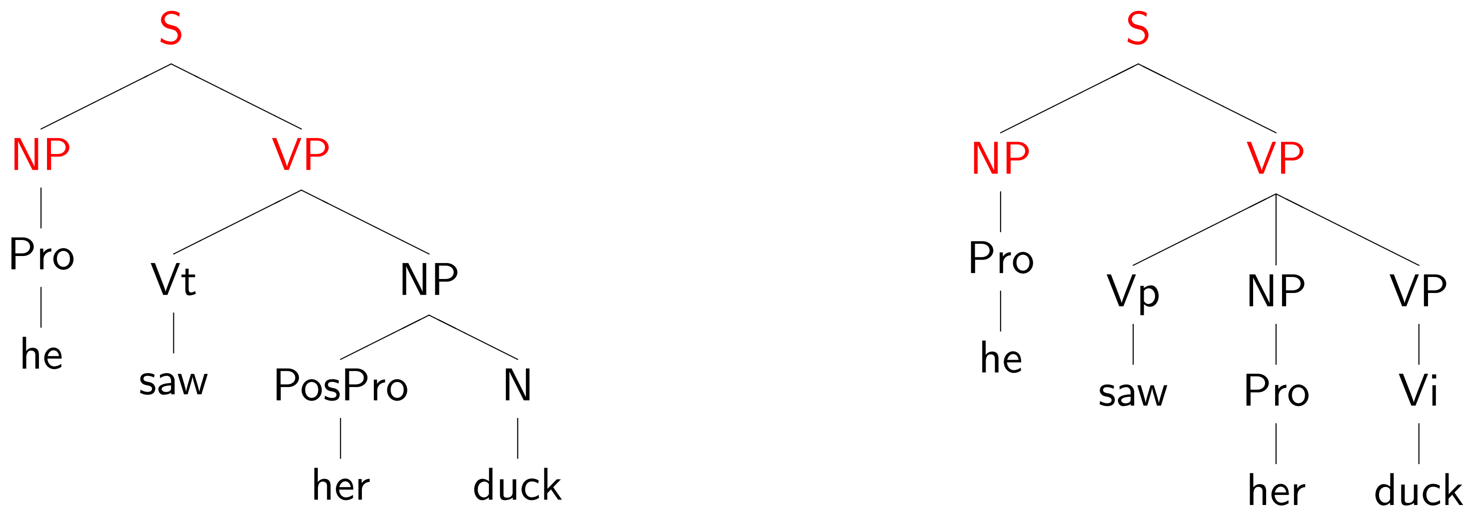
Compare **gold standard** tree (left) to **parser output** (right):



- Output constituent is counted correct if there is a gold constituent that spans the same sentence positions.
- Harsher measure: also require the constituent labels to match.
- **Pre-terminals** (lexical categories) don't count as constituents.

Evaluating parse accuracy

Compare **gold standard** tree (left) to **parser output** (right):



- **Precision**: (# correct constituents)/(# in parser output) = 3/5
- **Recall**: (# correct constituents)/(# in gold standard) = 3/4
- **F-score**: balances precision/recall: 2pr/(p+r)

Parsing accuracies

F-scores for parsing on WSJ corpus:

- vanilla PCFG: $< 80\%$[1]
- lexicalizing $+$ cat-splitting: 89.5% (Charniak, 2000)
- Best current parsers get about 92%
- Numbers get better if we look at top 5 or top 10

However, results on other corpora and other languages are considerably lower. Definitely not a solved problem!

[1] Charniak (1996) reports 81% but using gold POS tags as input.

Summary

- Probabilistic models of syntax can help disambiguation and speed in broad-coverage parsing.
  - by computing the probabilities of each tree or sub-tree as the product of the rules in it, and choosing the best option(s).
- Treebanks provide training data for estimating rule probabilities.
- However, to do well, we need to be clever:
  - Standard categories in the treebank don't capture some important facts about language.
  - By creating more detailed categories, we can encode more information within the PCFG framework.

Recall Problem with Vanilla PCFGs
No lexical dependencies

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

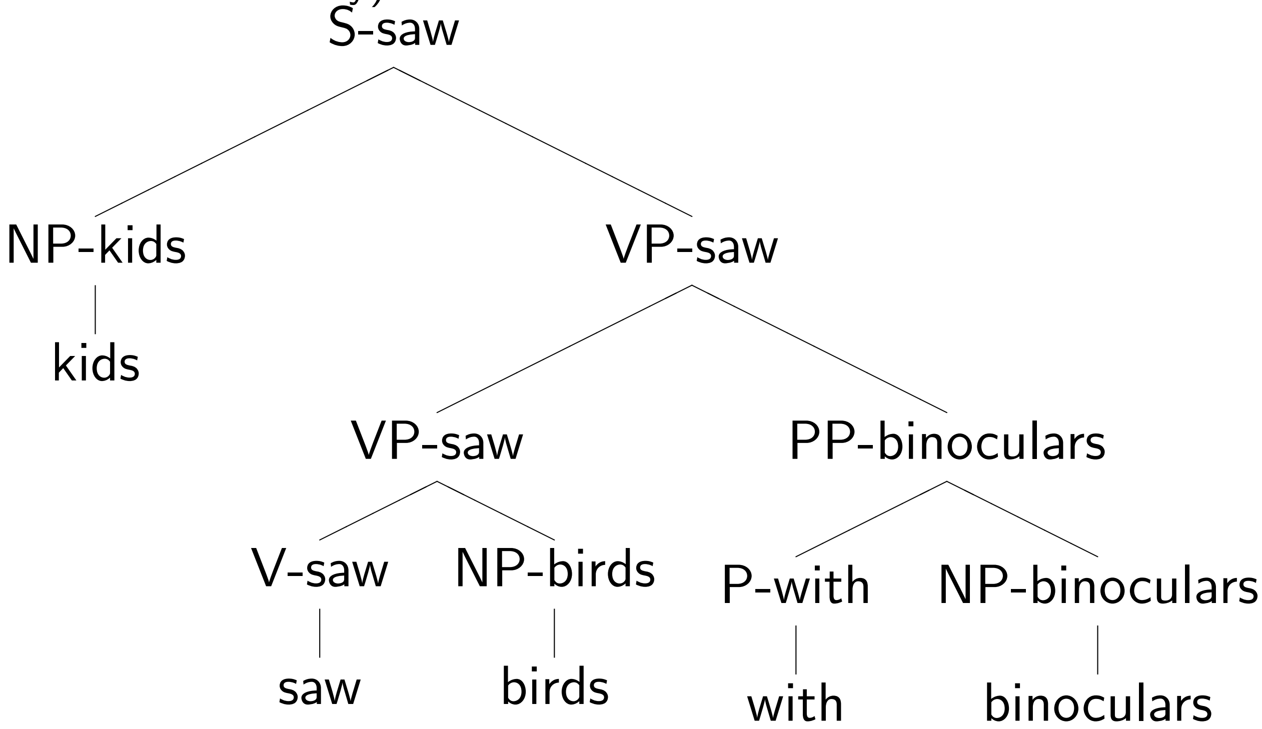
- kids saw birds with fish vs.
  kids saw birds with binoculars

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

- stray cats and dogs vs.
  Siamese cats and dogs

A way to fix PCFGs: lexicalization

Create new categories, this time by adding the **lexical head** of the phrase (note: N level under NPs not shown for brevity):



- Now consider:

`VP-saw` $\rightarrow$ `VP-saw PP-fish` vs. `VP-saw` $\rightarrow$ `VP-saw PP-binoculars`

Practical issues

- Identifying the head of every rule is not always straightforward
  - (more on this below)
- All this category-splitting makes the grammar much more **specific** (good!)
- But leads to huge grammar blowup and very sparse data (bad!)
- Lots of effort on how to balance these two issues.
  - Complex smoothing schemes (similar to N-gram interpolation/backoff).
  - More recently, increasing emphasis on automatically learned subcategories.
- But do we really need phrase structure in the first place? Not always!
- Today: Syntax (and parsing) without constituent structure.

Outline

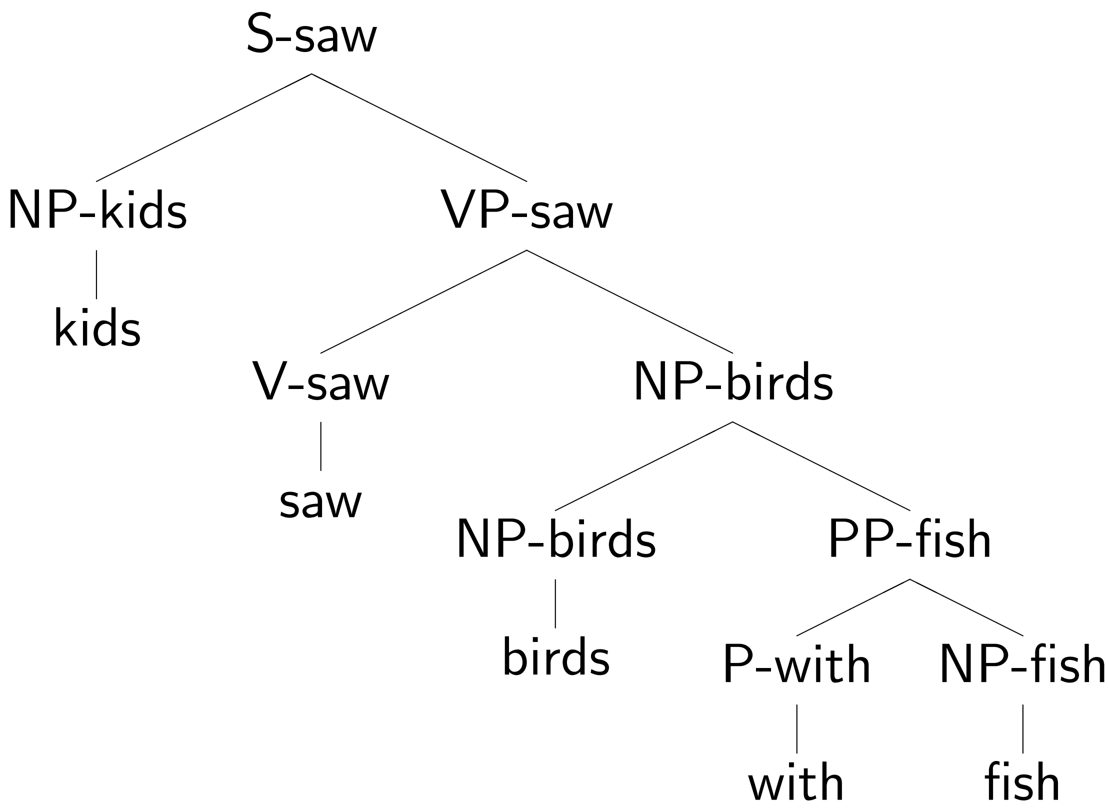
1. Dependencies: what/why

2. Transforming constituency $\rightarrow$ dependency parse

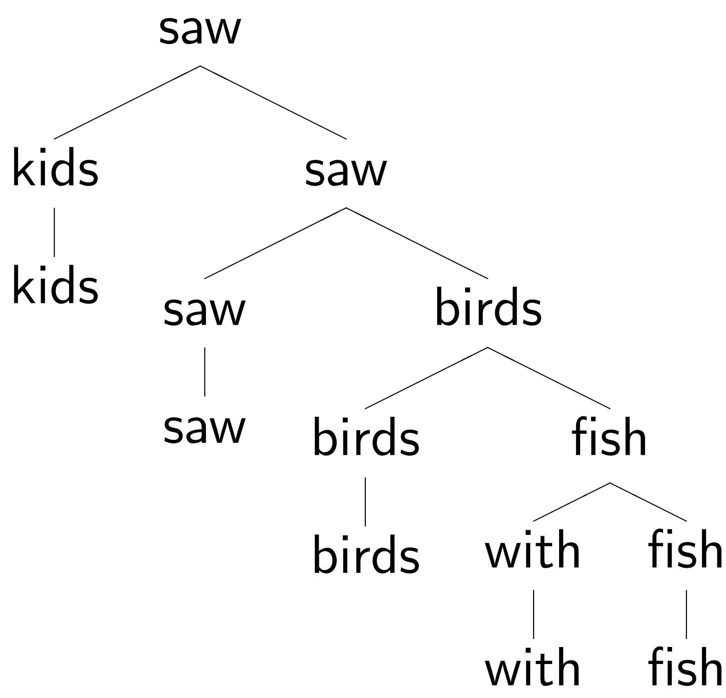
3. Direct dependency parsing

   - Transition-based (shift-reduce)
   - Graph-based

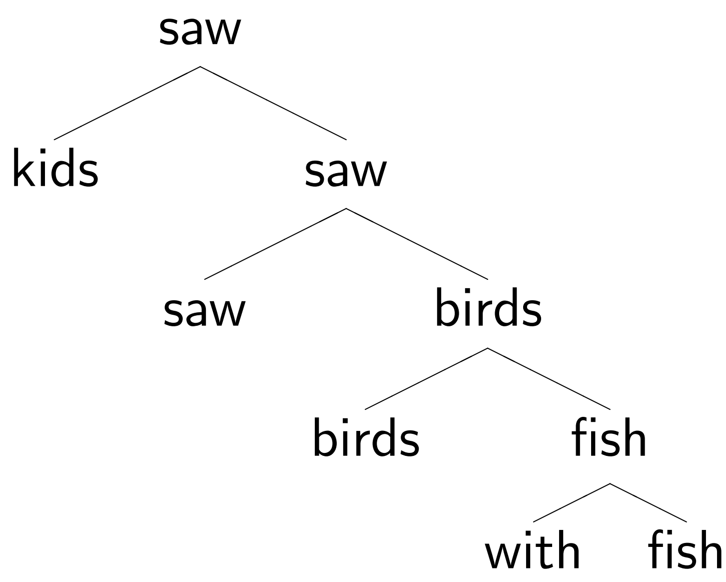
Lexicalized Constituency Parse



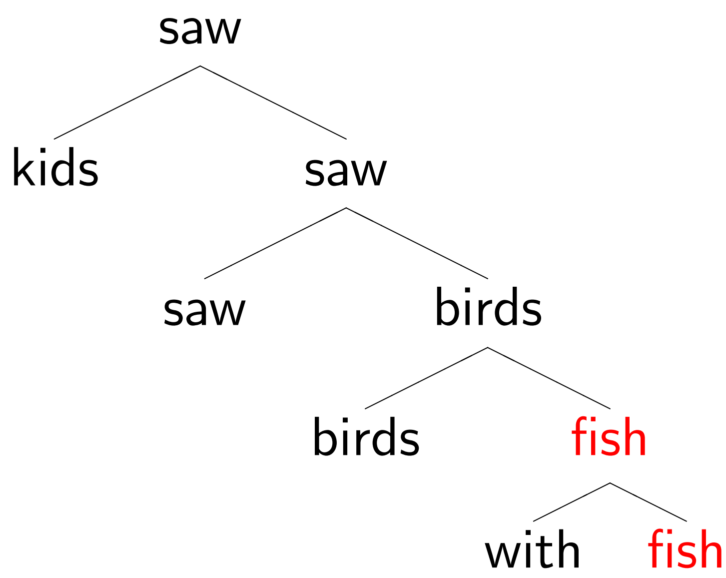
. . . remove the phrasal categories. . .



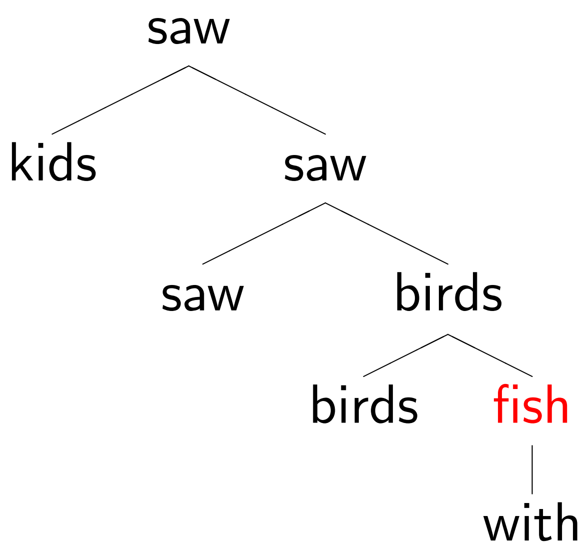
. . . remove the (duplicated) terminals. . .



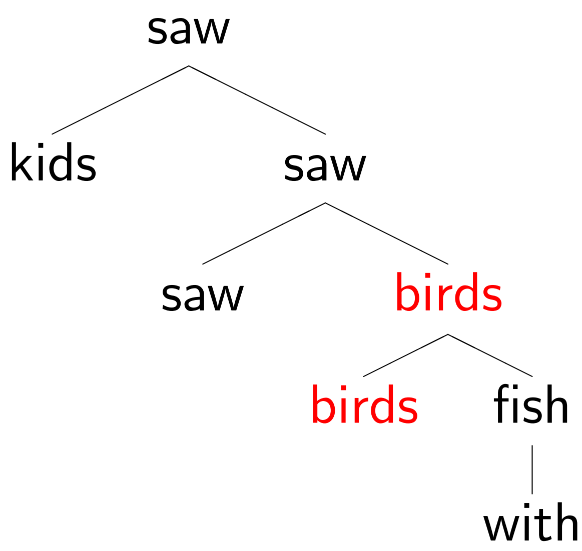
. . . and collapse chains of duplicates. . .



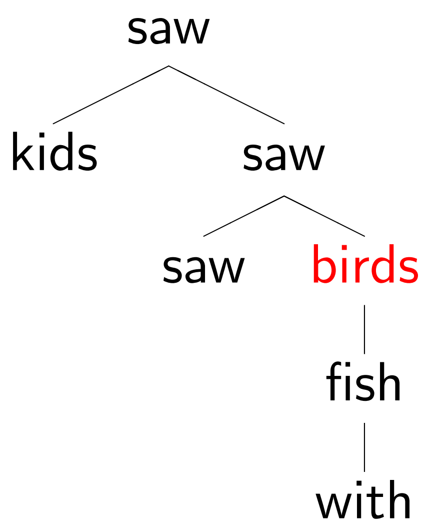
. . . and collapse chains of duplicates. . .



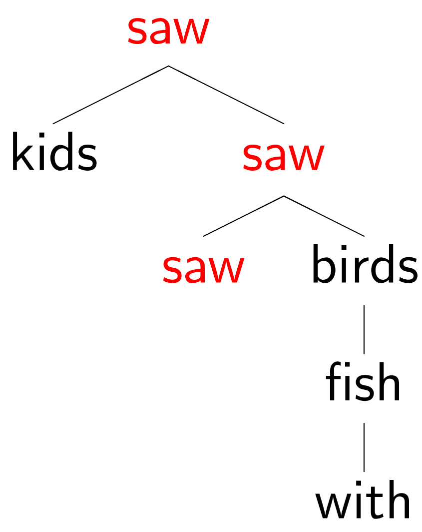
. . . and collapse chains of duplicates. . .



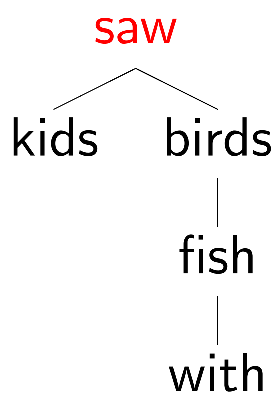
. . . and collapse chains of duplicates. . .



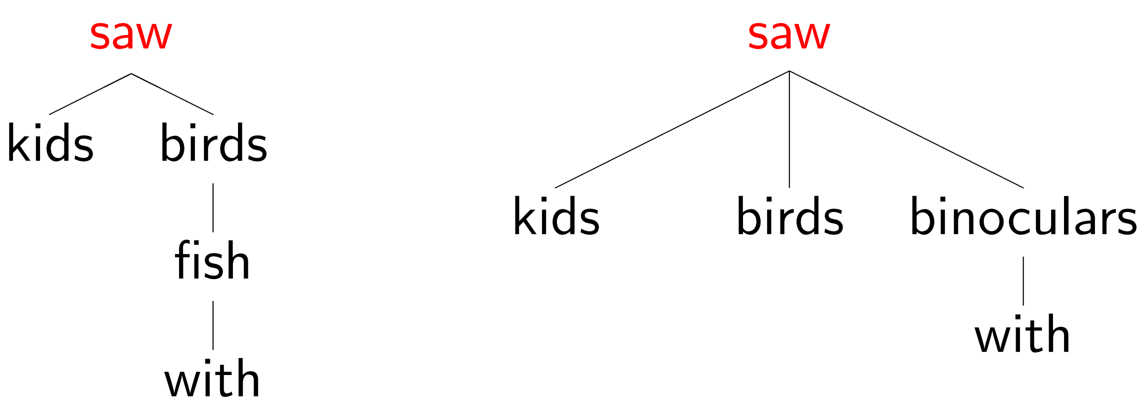
. . . and collapse chains of duplicates. . .



. . . and collapse chains of duplicates. . .



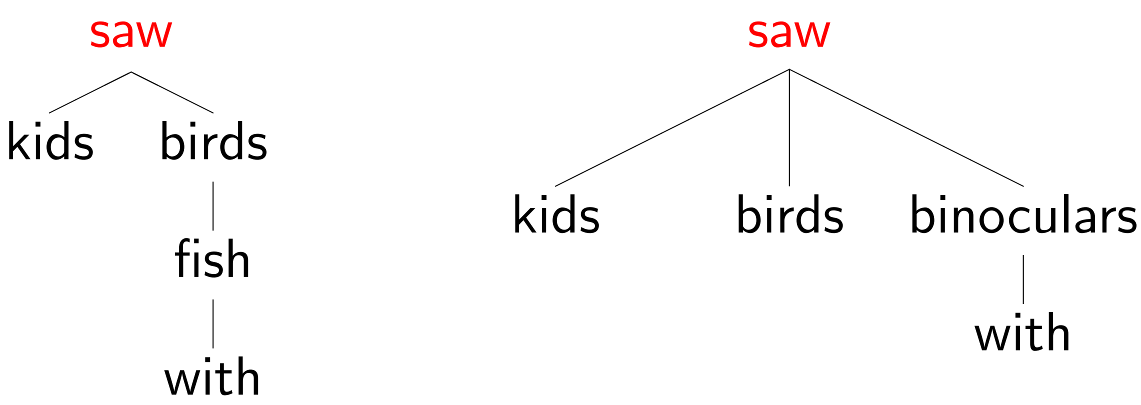
Dependency Parse



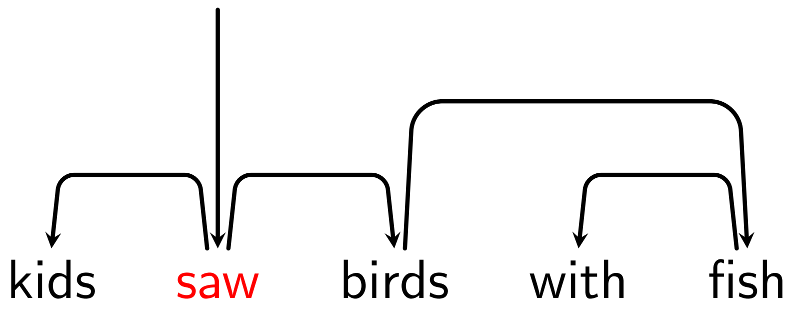
Linguists have long observed that the meanings of words within a sentence depend on one another, mostly in *asymmetric*, *binary* relations.

- Though some constructions don't cleanly fit this pattern: e.g., coordination, relative clauses.

Dependency Parse



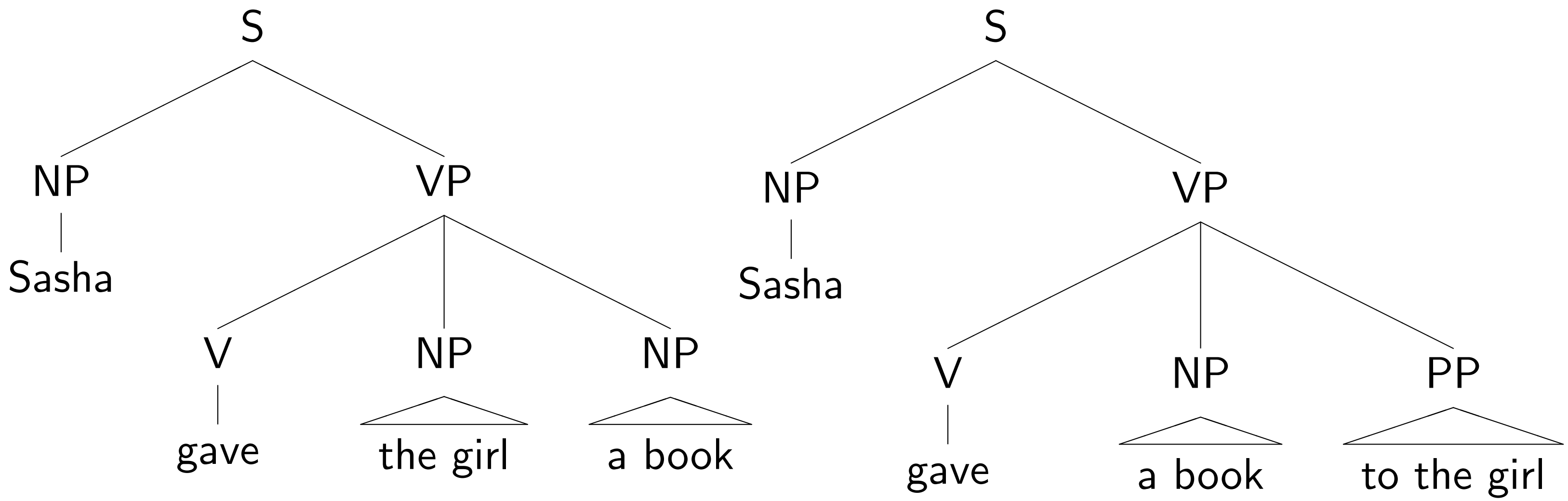
Equivalently, but showing word order (head $\rightarrow$ modifier):



Because it is a tree, every word has exactly one parent.

Why dependencies??

Consider these sentences. Two ways to say the same thing:



- We only need a few phrase structure rules:

  `S` $\rightarrow$ `NP VP`

  `VP` $\rightarrow$ `V NP NP`

  `VP` $\rightarrow$ `V NP PP`

  plus rules for NP and PP.

Equivalent sentences in Russian

- Russian uses morphology to mark relations between words:
  - knigu means book (kniga) as a direct object.
  - devochke means girl (devochka) as indirect object (to the girl).

- So we can have the same word orders as English:
  - Sasha dal devochke knigu
  - Sasha dal knigu devochke

Equivalent sentences in Russian

- Russian uses morphology to mark relations between words:
  - knigu means book (kniga) as a direct object.
  - devochke means girl (devochka) as indirect object (to the girl).
- So we can have the same word orders as English:
  - Sasha dal devochke knigu
  - Sasha dal knigu devochke
- But also many others!
  - Sasha devochke dal knigu
  - Devochke dal Sasha knigu
  - Knigu dal Sasha devochke

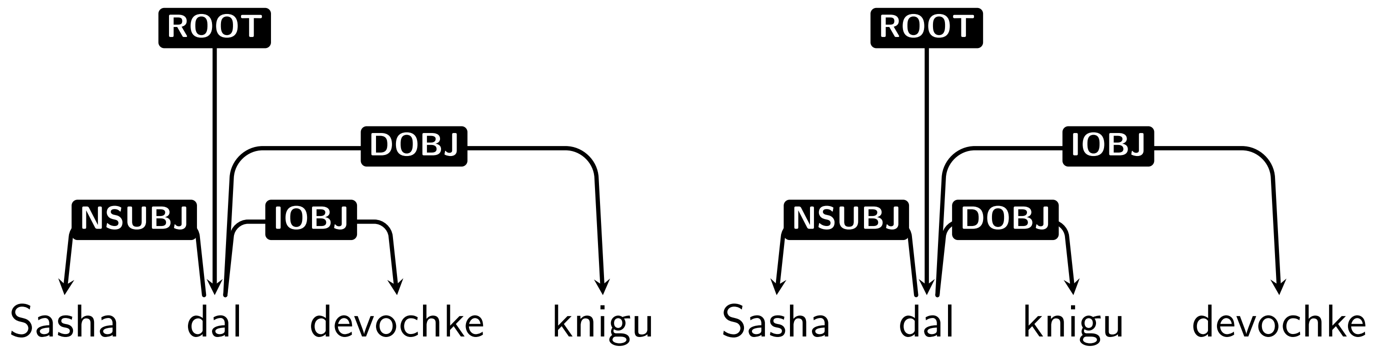
Phrase structure vs dependencies

- In languages with **free word order**, phrase structure (constituency) grammars don't make as much sense.
  - E.g., we would need both $\mathtt{S} \rightarrow \mathtt{NP}\ \mathtt{VP}$ and $\mathtt{S} \rightarrow \mathtt{VP}\ \mathtt{NP}$, etc. Not very informative about what's really going on.

Phrase structure vs dependencies

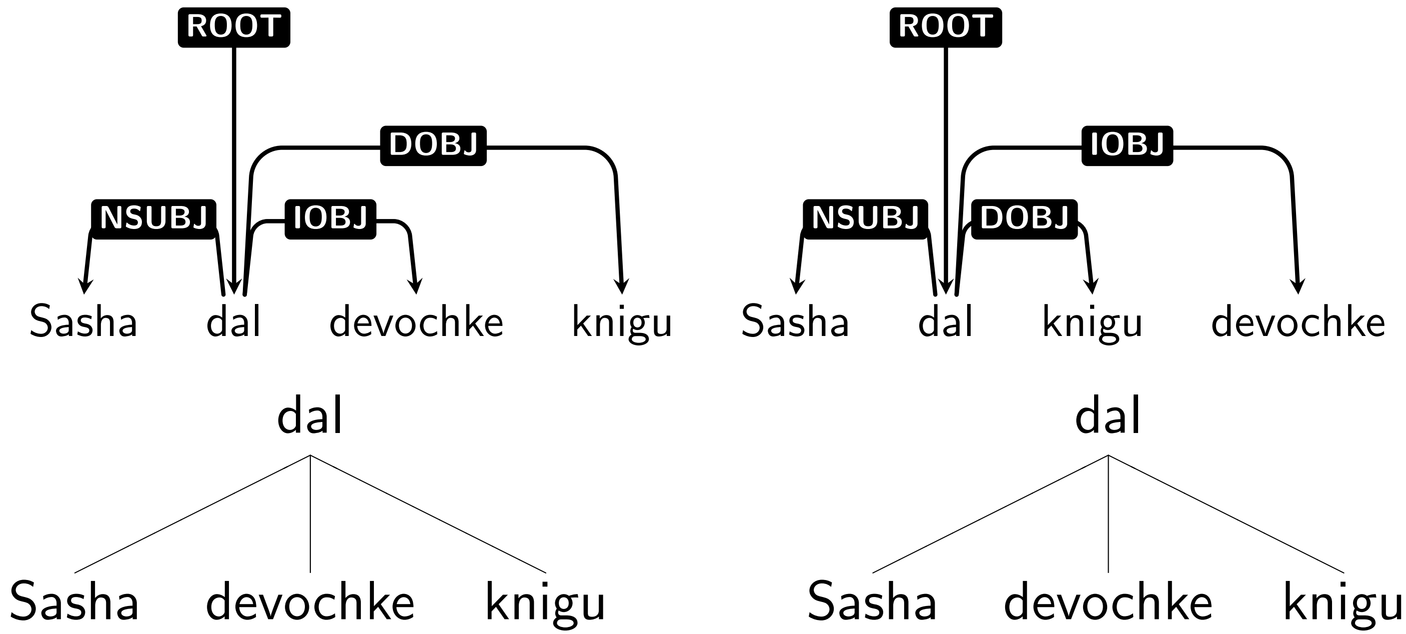
- In languages with **free word order**, phrase structure (constituency) grammars don't make as much sense.
  - E.g., we would need both $\texttt{S} \rightarrow \texttt{NP VP}$ and $\texttt{S} \rightarrow \texttt{VP NP}$, etc. Not very informative about what's really going on.

- In contrast, the dependency relations stay constant:



Phrase structure vs dependencies

- Even more obvious if we just look at the trees without word order:



Pros and cons

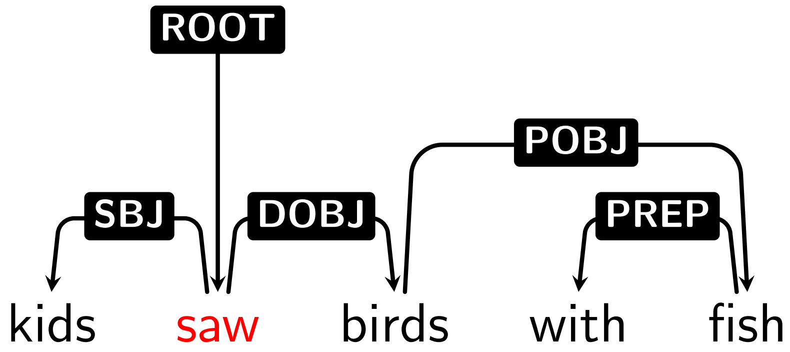
- Sensible framework for free word order languages.
- Identifies syntactic relations directly. (using CFG, how would you identify the subject of a sentence?)
- Dependency pairs/chains can make good features in classifiers, for information extraction, etc.
- Parsers can be very fast (coming up...)

But

- The assumption of asymmetric binary relations isn't always right... e.g., how to parse dogs and cats?

Edge Labels

It is often useful to distinguish different kinds of head $\rightarrow$ modifier **relations**, by labeling edges:

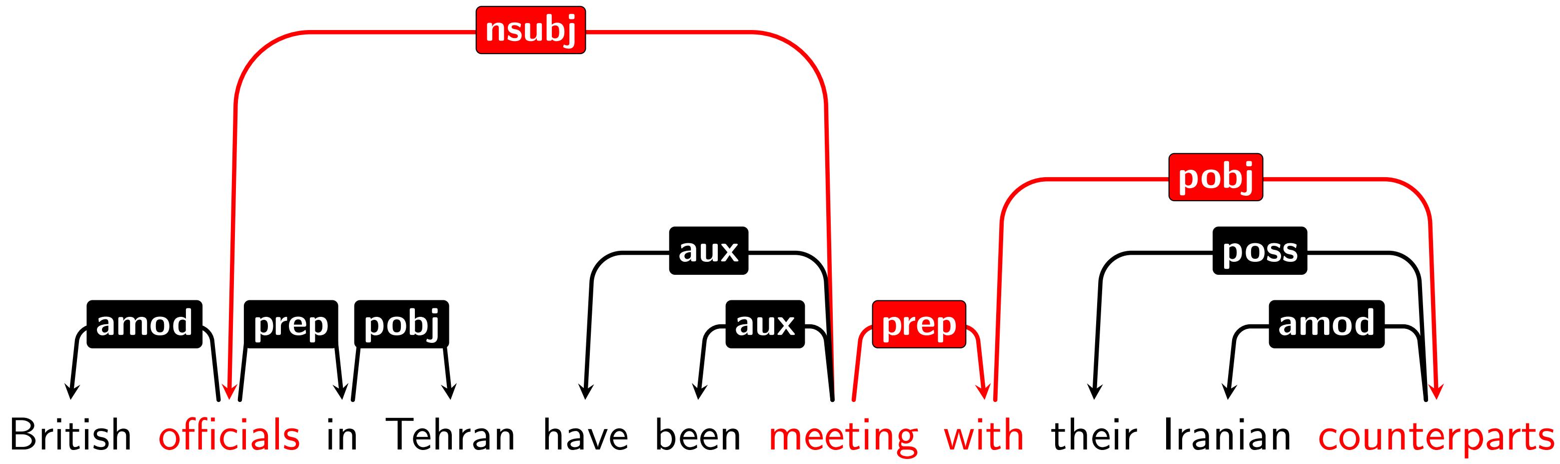


Important relations for English include subject, direct object, determiner, adjective modifier, adverbial modifier, etc. (Different treebanks use somewhat different label sets.)

- How would you identify the subject in a constituency parse?

Dependency Paths

For **information extraction** tasks involving real-world relationships between entities, chains of dependencies can provide good features:

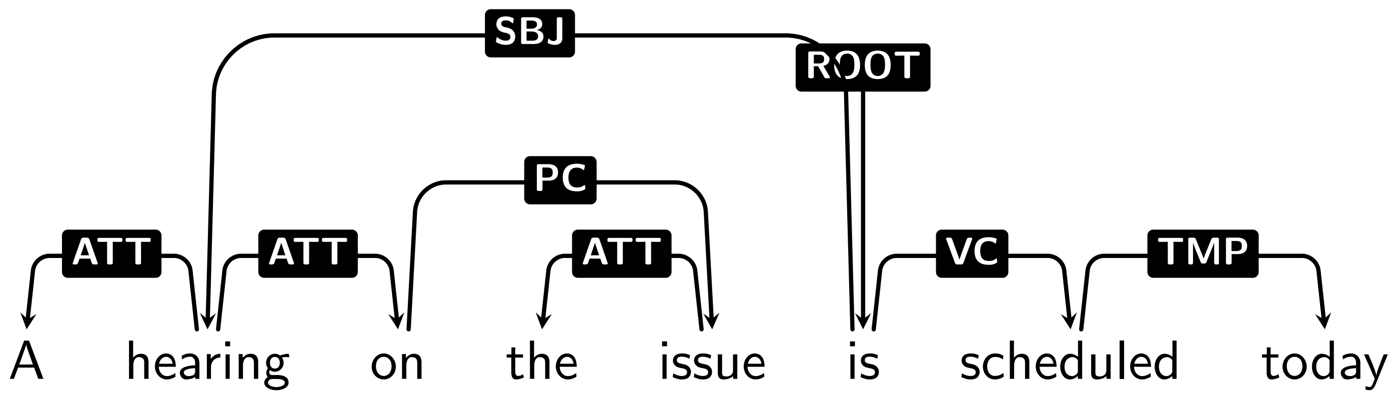


(example from Brendan O'Connor)

Projectivity

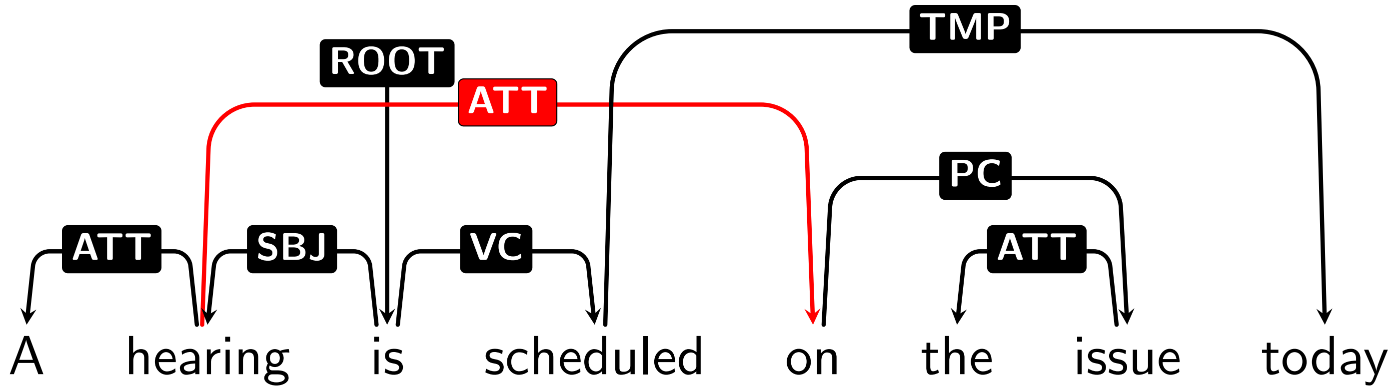
- A sentence's dependency parse is said to be **projective** if every subtree (node and all its descendants) occupies a *contiguous span* of the sentence.

- = The dependency parse can be drawn on top of the sentence without any crossing edges.



Nonprojectivity

- Other sentences are **nonprojective**:



- Nonprojectivity is rare in English, but quite common in many languages.

Outline

1. Dependencies: what/why

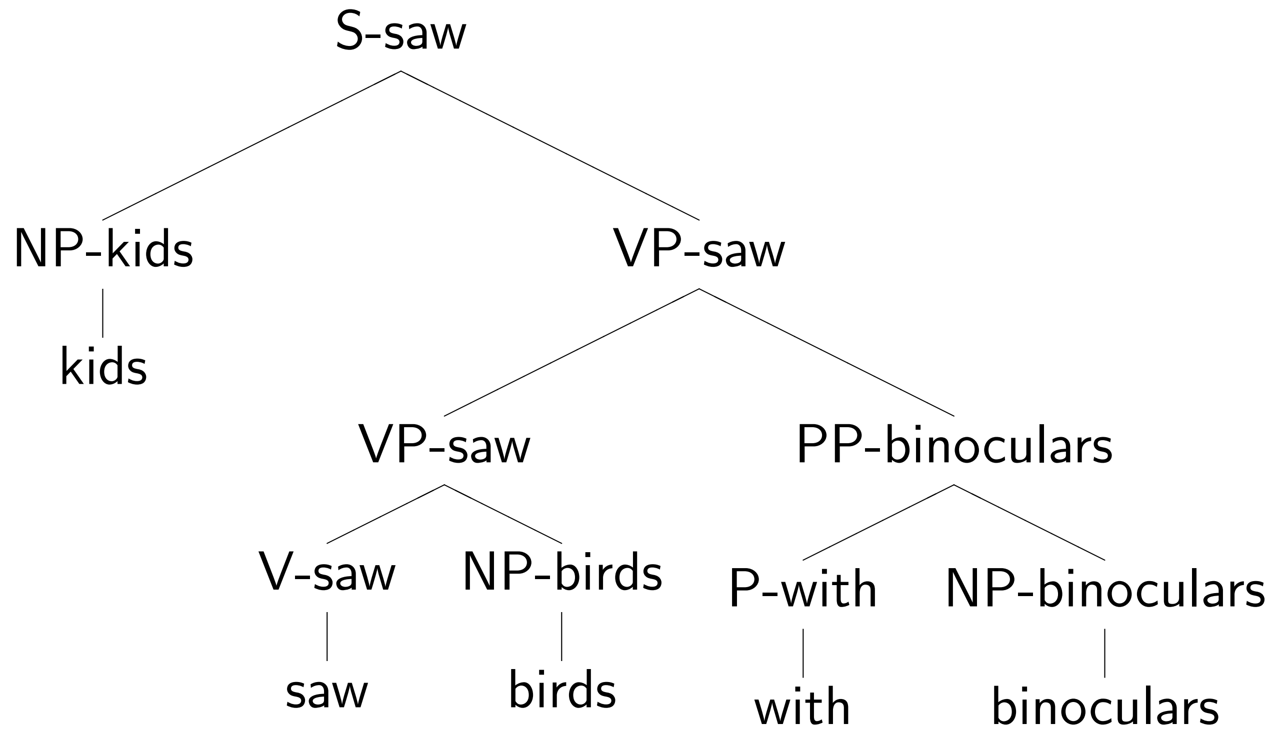
2. **Transforming constituency $\rightarrow$ dependency parse**

3. Direct dependency parsing

   - Transition-based (shift-reduce)
   - Graph-based

Constituency Tree → Dependency Tree

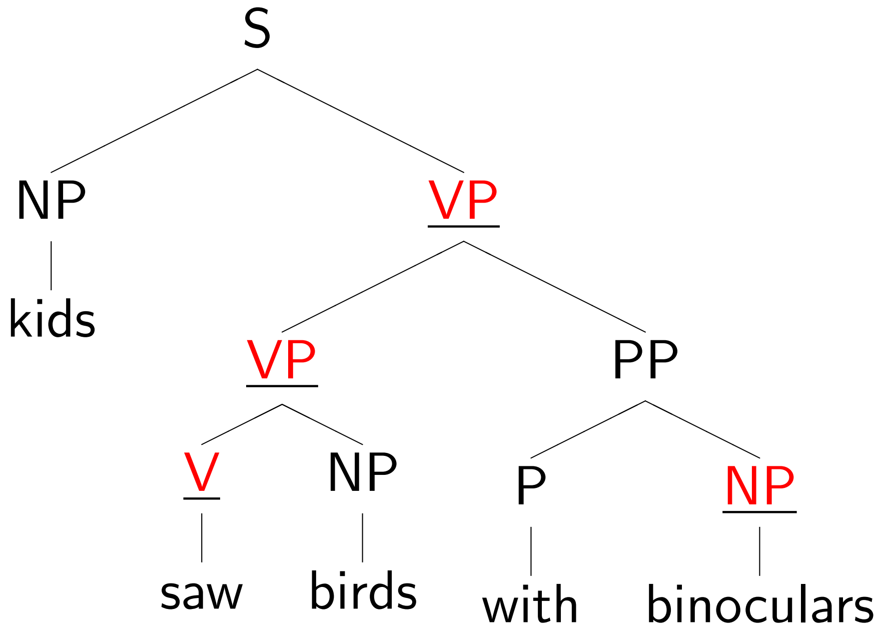
We saw how the **lexical head** of the phrase can be used to collapse down to a dependency tree:



- But how can we find each phrase's head in the first place?

Head Rules

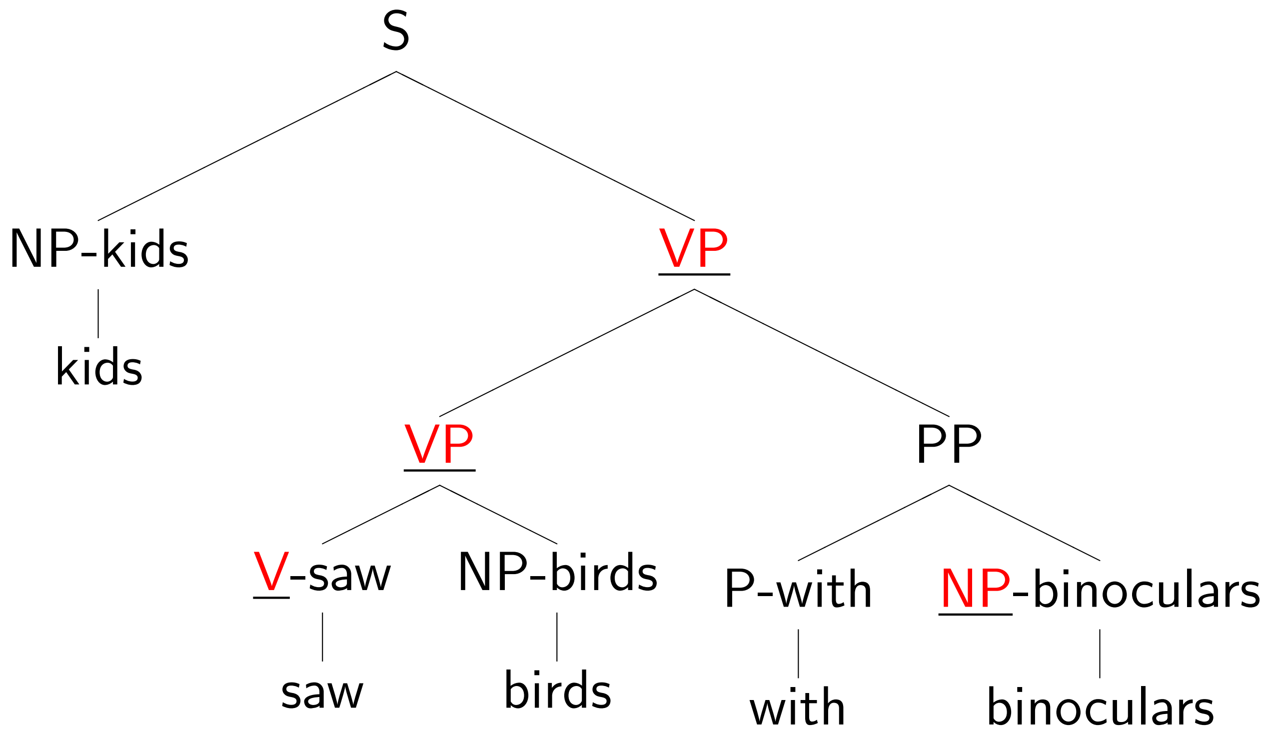
The standard solution is to use **head rules**: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. $\mathtt{S} \rightarrow \mathtt{NP}\ \underline{\mathtt{VP}}$, $\mathtt{VP} \rightarrow \underline{\mathtt{VP}}\ \mathtt{PP}$, $\mathtt{PP} \rightarrow \mathtt{P}\ \underline{\mathtt{NP}}$, etc.



- Heuristics to scale this to large grammars: e.g., within an NP, last immediate N child is the head.

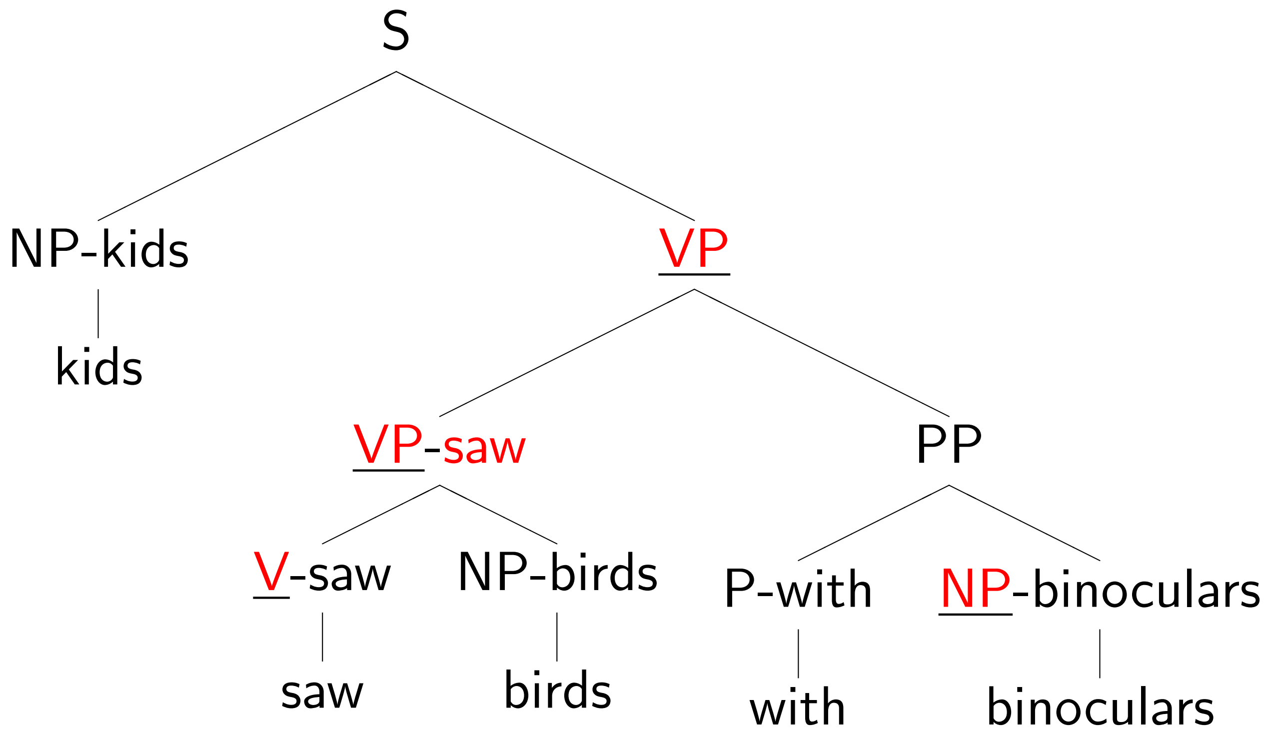
Head Rules

Then, propagate heads up the tree:



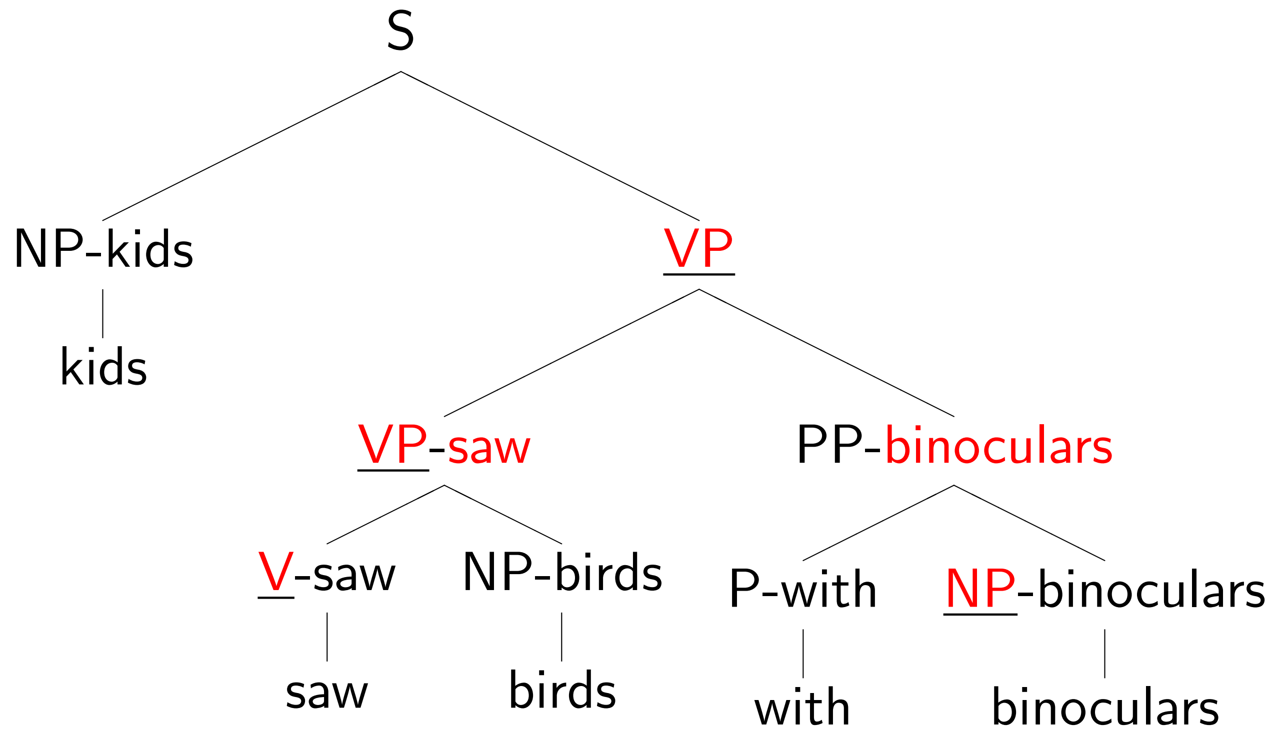
Head Rules

Then, propagate heads up the tree:



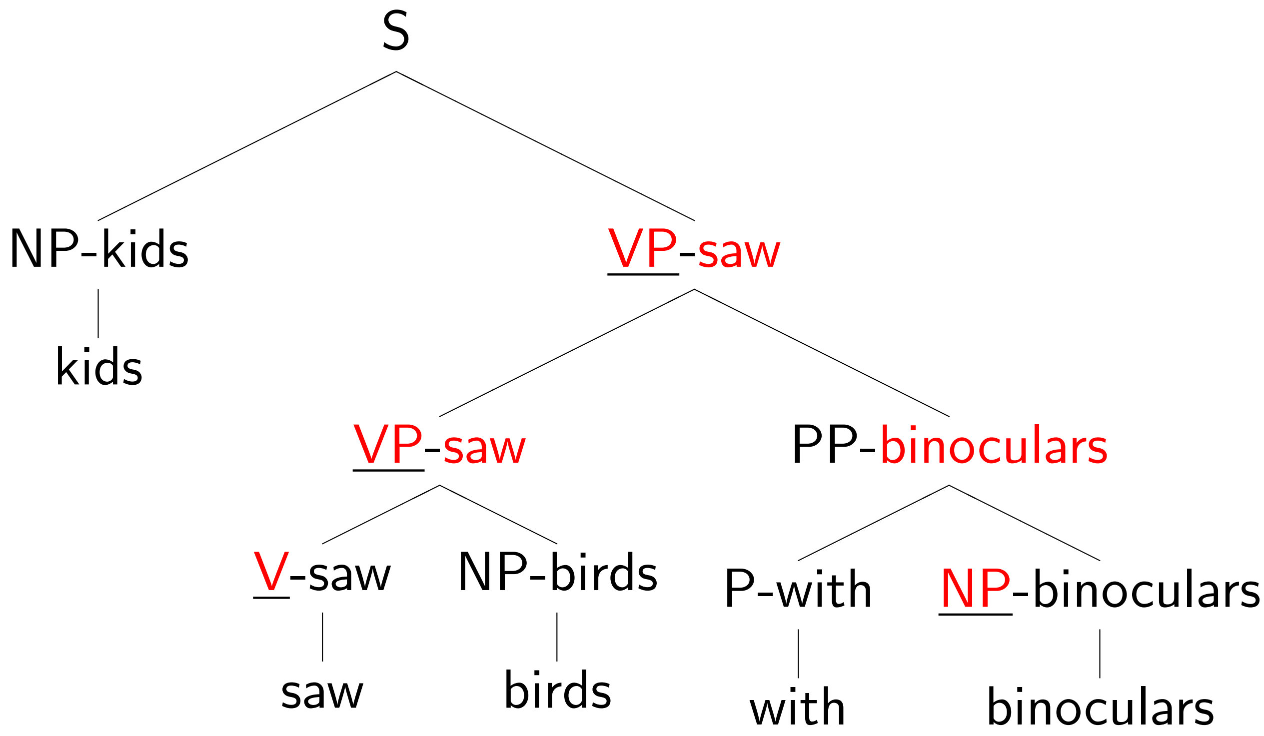
Head Rules

Then, propagate heads up the tree:



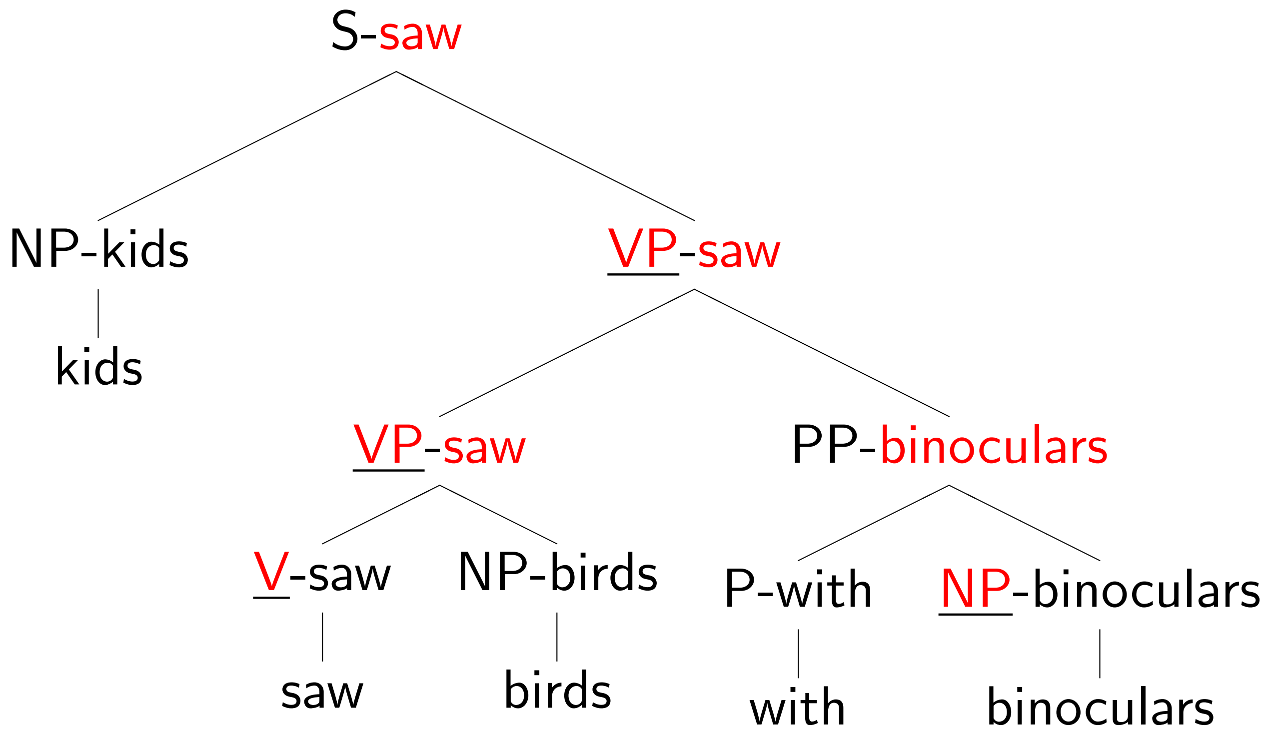
Head Rules

Then, propagate heads up the tree:



Head Rules

Then, propagate heads up the tree:



Outline

1. Dependencies: what/why

2. Transforming constituency $\rightarrow$ dependency parse

3. **Direct dependency parsing**

   - Transition-based (shift-reduce)
   - Graph-based

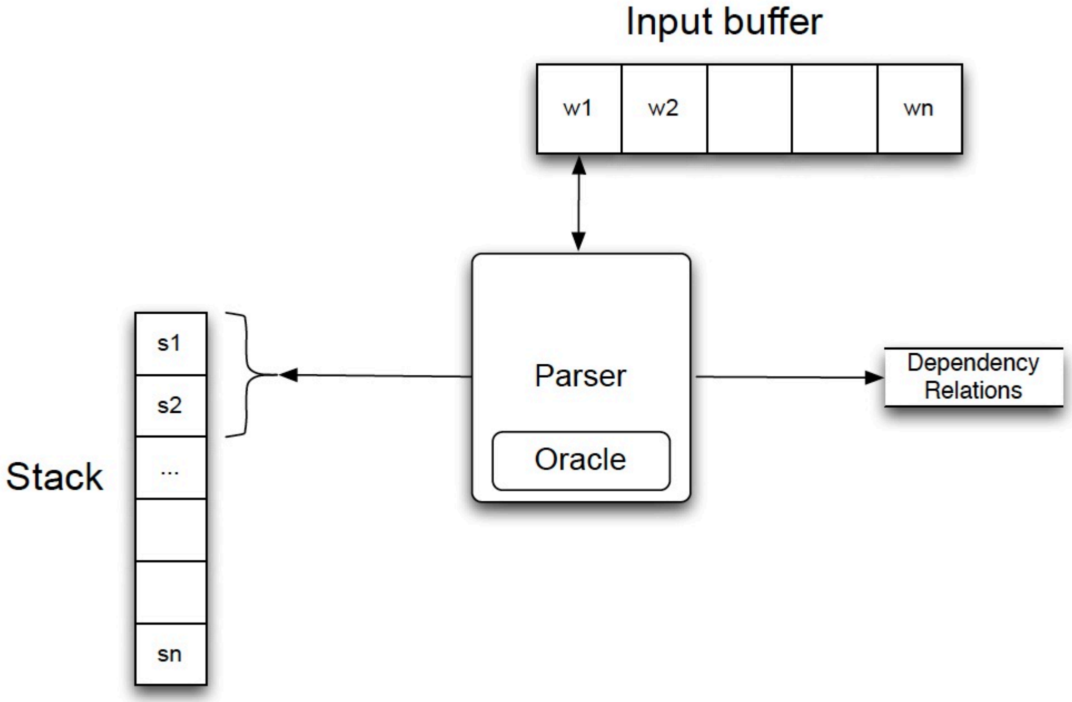
Dependency Parsing

Some of the algorithms you have seen for PCFGs can be adapted to dependency parsing.

- **CKY** can be adapted, though efficiency is a concern: obvious approach is $O(Gn^5)$; Eisner algorithm brings it down to $O(Gn^3)$
  - N. Smith's slides explaining the Eisner algorithm: `http://courses.cs.washington.edu/courses/cse517/16wi/slides/an-dep-slides.pdf`

- **Shift-reduce**: more efficient, doesn't even require a grammar!

Transitation-based Parsing: Shift Reduce Parser



3 possible actions:

LeftArc: Assign head-dependent relation between $s1$ and $s2$; pop $s2$

RightArc: Assign head-dependent relation between $s2$ and $s1$; pop $s1$

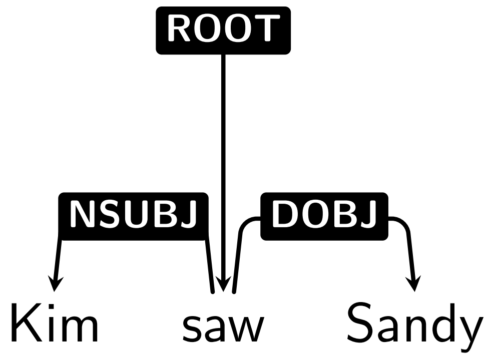
Shift: Put $w1$ on top of the stack.

Remember, dependency relation points *from* head *to* dependent

Both **LeftArc** and **RightArc** pop the dependent, leaving the head at the top of the stack

Example

Step	Stack	Word List	Action	Relations
0	[root]	[Kim,saw,Sandy]		
1	[root,Kim]	[saw,Sandy]	Shift	
2	[root,Kim,saw]	[Sandy]	Shift	
3	[root,saw]	[Sandy]	LeftArc	nsubj(saw,Kim)
4	[root,saw,Sandy]	[]	Shift	
5	[root,saw]	[]	RightArc	dobj(saw,Sandy)
6	[root]	[]	RightArc	root→saw



Transition-based Parsing

- Latent structure is just edges between words. Train a **classifier** as the oracle to predict next action (Shift, LeftArc, or RightArc), and proceed left-to-right through the sentence. $O(n)$ time complexity!

- Only finds **projective** trees (without special extensions)

- Pioneering system: Nivre's MaltParser

- See `http://spark-public.s3.amazonaws.com/nlp/slides/Parsing-Dependency.pdf` (Jurafsky & Manning Coursera slides) for details and examples

Graph-based Parsing

- Global algorithm: From the fully connected directed graph of all possible edges, choose the best ones that form a tree.

- **Edge-factored** models: Classifier assigns a nonnegative score to each possible edge; **maximum spanning tree** algorithm finds the spanning tree with highest total score in $O(n^2)$ time.
  - Edge-factored assumption can be relaxed (higher-order models score larger units; more expensive).
  - Unlabeled parse $\rightarrow$ edge-labeling classifier (pipeline).

- Pioneering work: McDonald's MSTParser

- Can be formulated as constraint-satisfaction with **integer linear programming** (Martins's TurboParser)

Graph-based vs. Transition-based vs. Conversion-based

- TB: Features in scoring function can look at any part of the stack; no optimality guarantees for search; linear-time; (classically) projective only
- GB: Features in scoring function limited by factorization; optimal search within that model; quadratic-time; no projectivity constraint
- CB: In terms of accuracy, sometimes best to first constituency-parse, then convert to dependencies (e.g., Stanford Parser).
  - Slower than direct methods.
  - And, you need a grammar and head rules.

Choosing a Parser: Criteria

- Target representation: constituency or dependency?
- Efficiency? In practice, both runtime and memory use.
- Incrementality: parse the whole sentence at once, or obtain partial left-to-right analyses/expectations?
- Retrainable system?

Summary

- While constituency parses give hierarchically nested phrases, dependency parses represent syntax with trees whose edges connect words in the sentence. (No abstract phrase categories like `NP`.) Edges often labeled with relations like subject.

- Head rules govern how a lexicalized constituency grammar can be extracted from a treebank, and how a constituency parse can be coverted to a dependency parse.

- For English, it is often fastest and most convenient to parse directly to dependencies. Two main paradigms, graph-based and transition-based, with different kinds of models and search algorithms.