Foundations of Natural Language Processing
Lecture 17
Heads, Dependency parsing

Ivan Titov
(slides from Alex Lascarides, Henry Thompson, Nathan Schneider and Sharon Goldwater)

29 February 2024

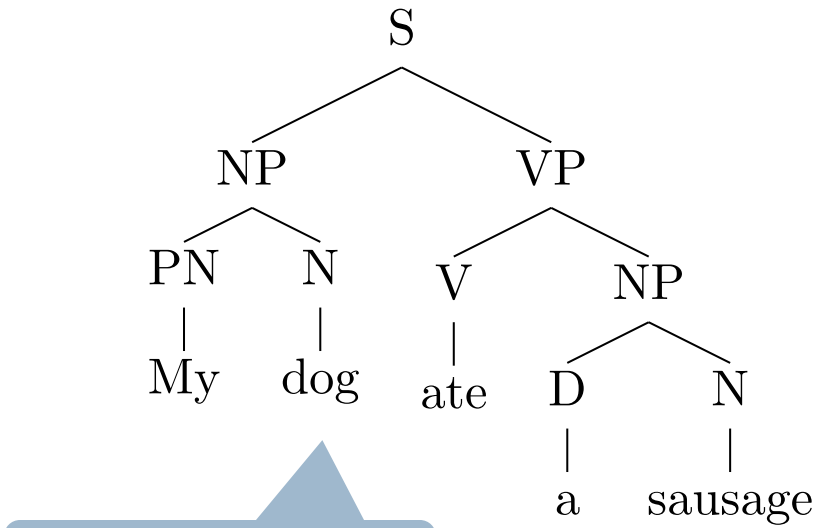
School of informatics

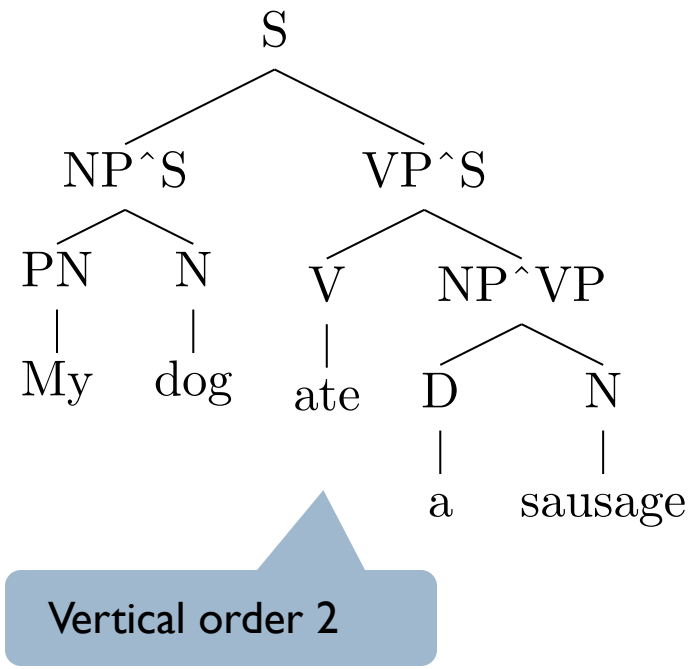
Recap: Weaknesses of (treebank) PCFGs

- They do not encode lexical preferences
- They do not encode structural properties (beyond single rules)

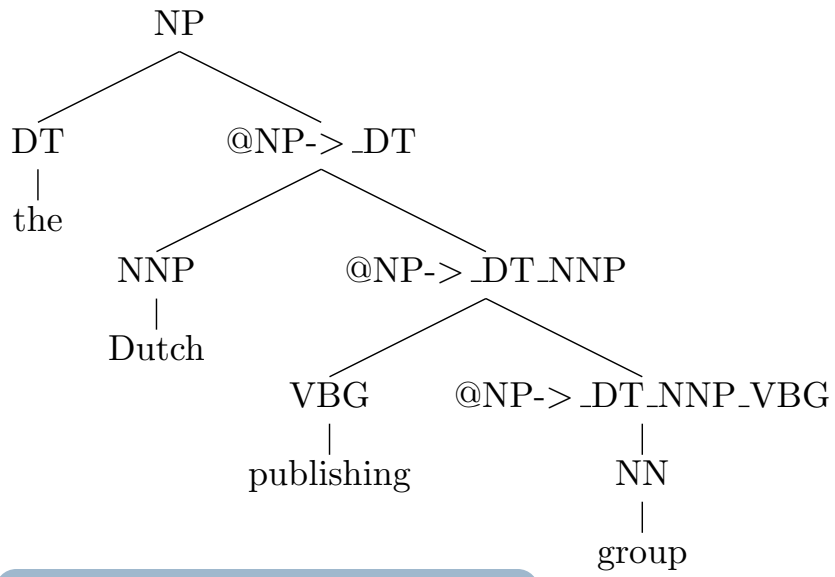
Recap: Vertical Markovization

- Rule applications depend on past ancestors in the tree (not only parents) *[Johnson 98]*

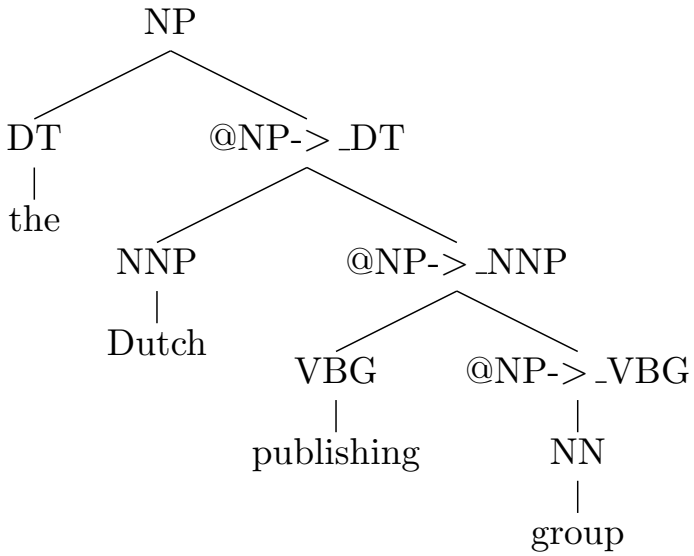




Recap: Horizontal Markovization

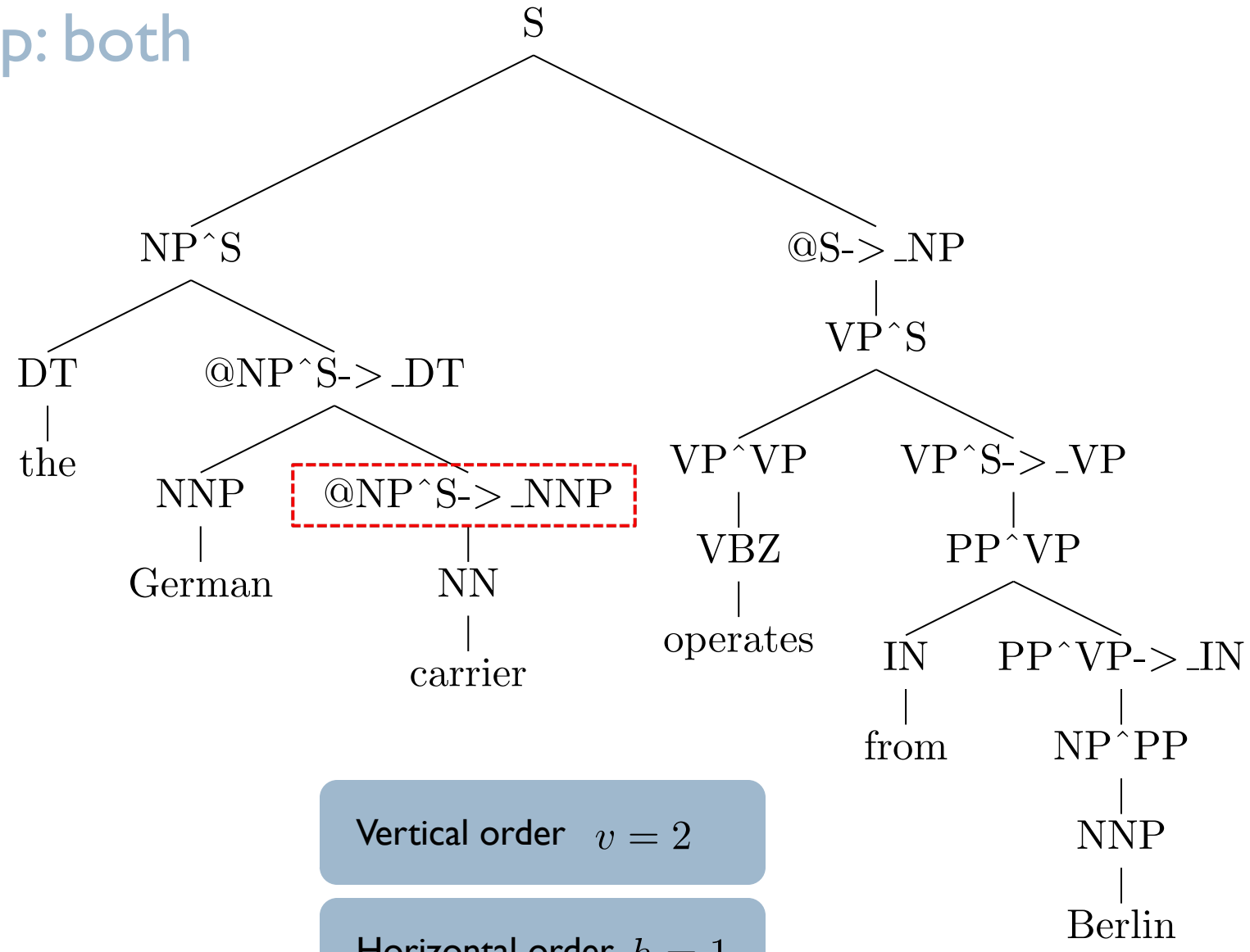


Horizontal order $h = \infty$



Horizontal order $h = 1$

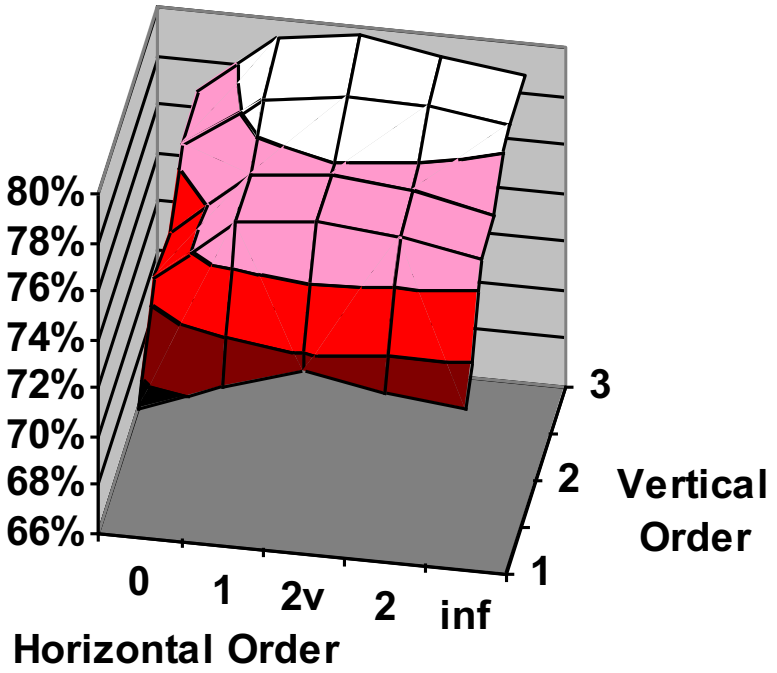
Recap: both



Vertical order $v = 2$

Horizontal order $h = 1$

Vertical and Horizontal Markovization



Around 78%, compare with 72% for the original treebank PCFG

Splitting: PoS tags

- PoS tags in Penn Treebank are too coarse
- Very obvious for IN tag:
  - Assigned both to 'normal' prepositions (to form a prepositional phrase) – *in, on, at, …* –
  - and to subordinating conjunctions (e.g., *if)*
    - *E.g., check if advertising works*
- This change alone leads to a 2% boost in performance:
  - from 78.2 to 80.3

[Klein and Manning 2003]

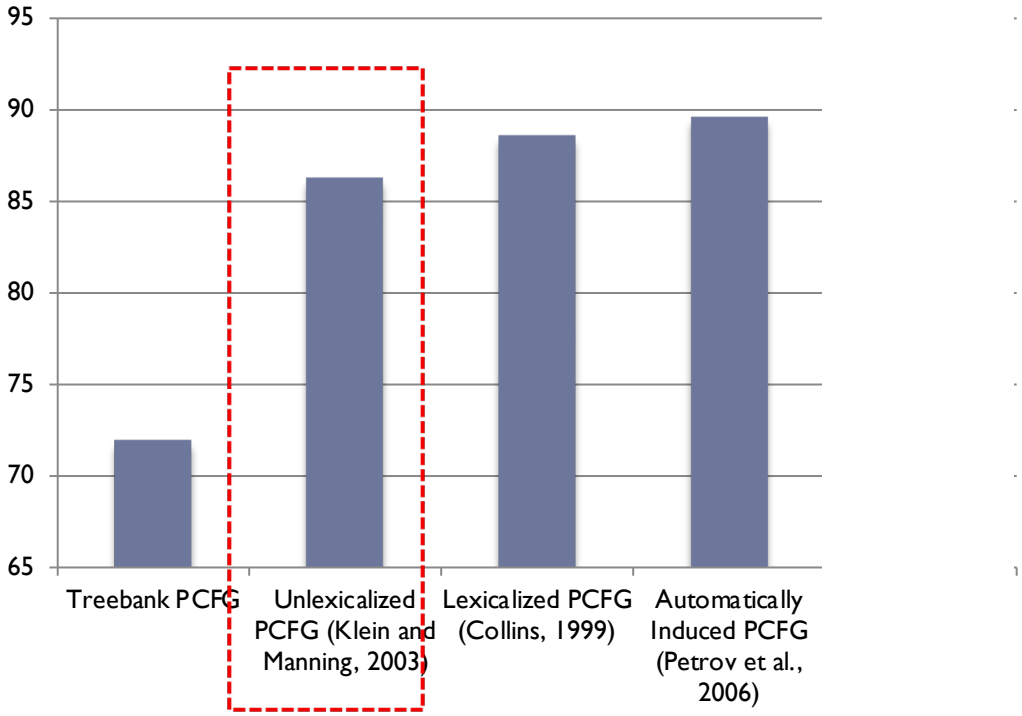
Splitting: other symbols

- Split determiners: on demonstrative ("those") and others (e.g., "the", "a")
- Split adverbials: on phrasal and not ("quickly" vs. "very")
- …

All these changes (and a couple of other ones) lead to 86.3 % F1, a very respectable (and maybe even surprising) performance for an unlexicalized PCFG model

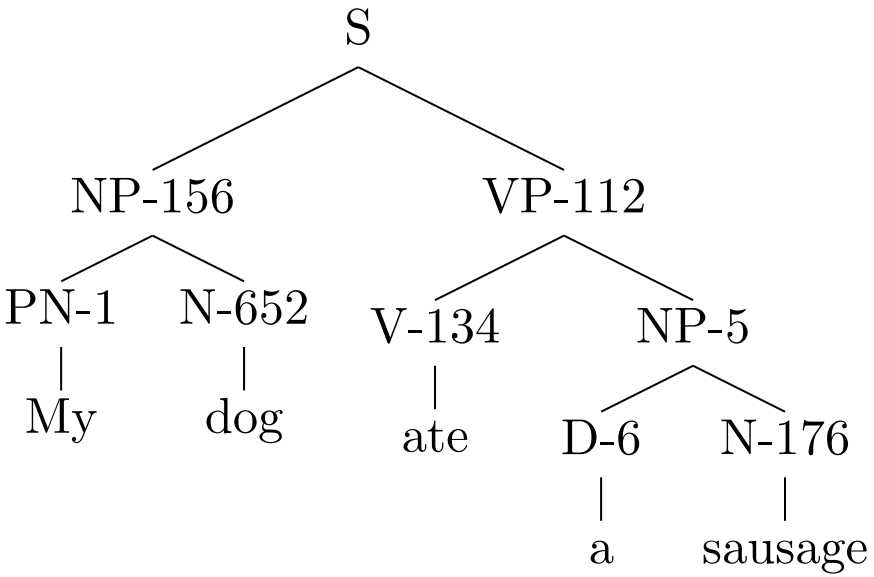
[Klein and Manning 2003]

Preview: F1 bracket score



Alternative ideas: inducing splits (through EM)

- Learning types of nonterminals from data, i.e. automatically enriching the grammar (Latent-annotated PCFGs, LA-PCFG)
  - One can think of this as a type of clustering of tree contexts of non-terminal symbols



Around 90% F1

[Matsuzaki et al., 2005, Petrov et al., 2006]

Alternative ideas: anchored rules

- A rule probability is not constant but predicting for a given span in the chart
  - E.g., a neural network predicts the probability of a rule for a specific operation of the chart

```
double t1 = chart[min][mid][C1]

double t2 = chart[mid][max][C2]

double candidate = t1 * t2 * p(C -> C1 C2)
```

Instead use:

$$t_1 * t_2 * NN_\theta(C_1, C_2, C_3, min, max, mid, \mathbf{x})$$

Up to 97% F1

First in Cross and Huang (2016)

Summary for PCFG parsing

- PCFGs for statistical parsing
- Dynamic programming algorithm for parsing with PCFGs
- Vanilla treebank PCFGs parser is (very) weak
- … but can be improved to produce a very strong system
- CKY is an important tool, used in many applications

Summary

- PCFGs for statistical parsing
- Dynamic programming algorithm for parsing with PCFGs
- Vanilla treebank PCFGs parser is (very) weak
- … but can be improved to produce a very strong system
- CKY is an important tool, used in many applications

Today we will. . .

- Return to the problem of PCFGs
- Look in detail into 'lexicalization' we mentioned last time
- This fix leads to an approach without constituent structure!

Dependency parsing

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

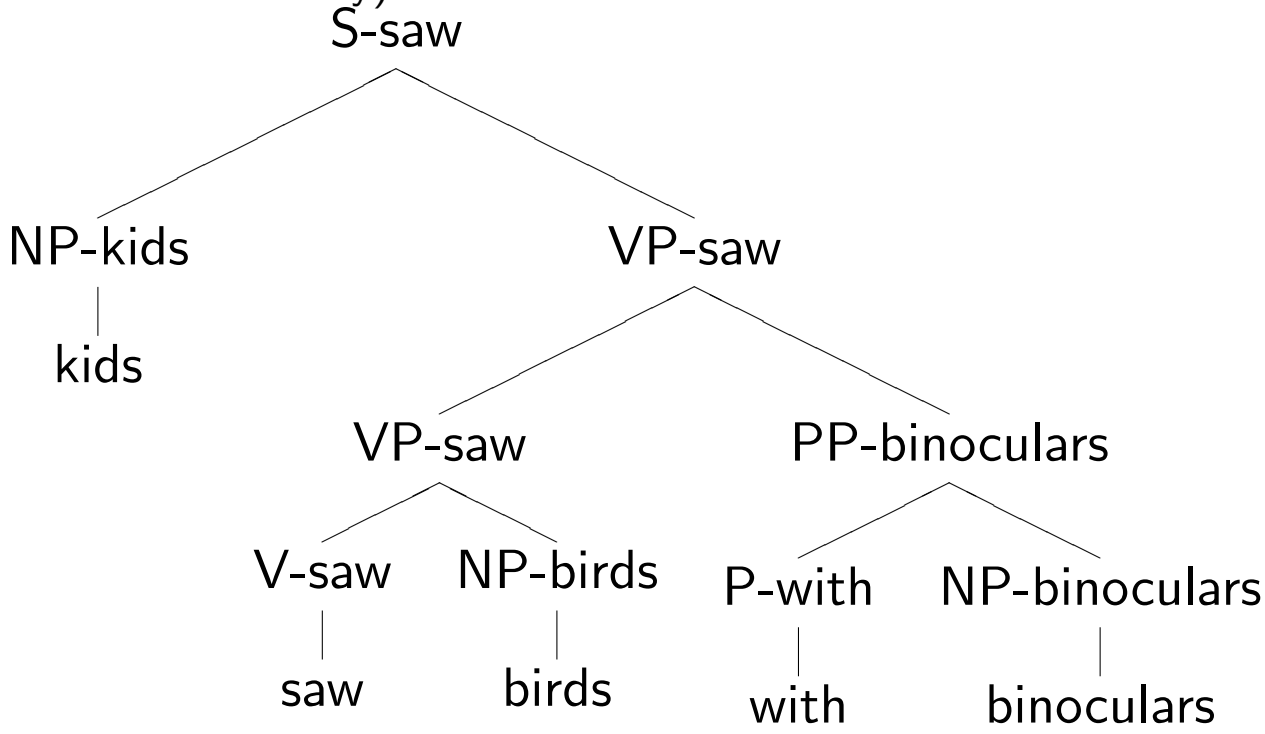
- She stood by the door covered in tears vs.
  She stood by the door covered in ivy

- stray cats and dogs vs.
  Siamese cats and dogs

We discussed techniques which produce or induces splits of POS tags, but this is often not enough (as in the examples above)

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

- 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

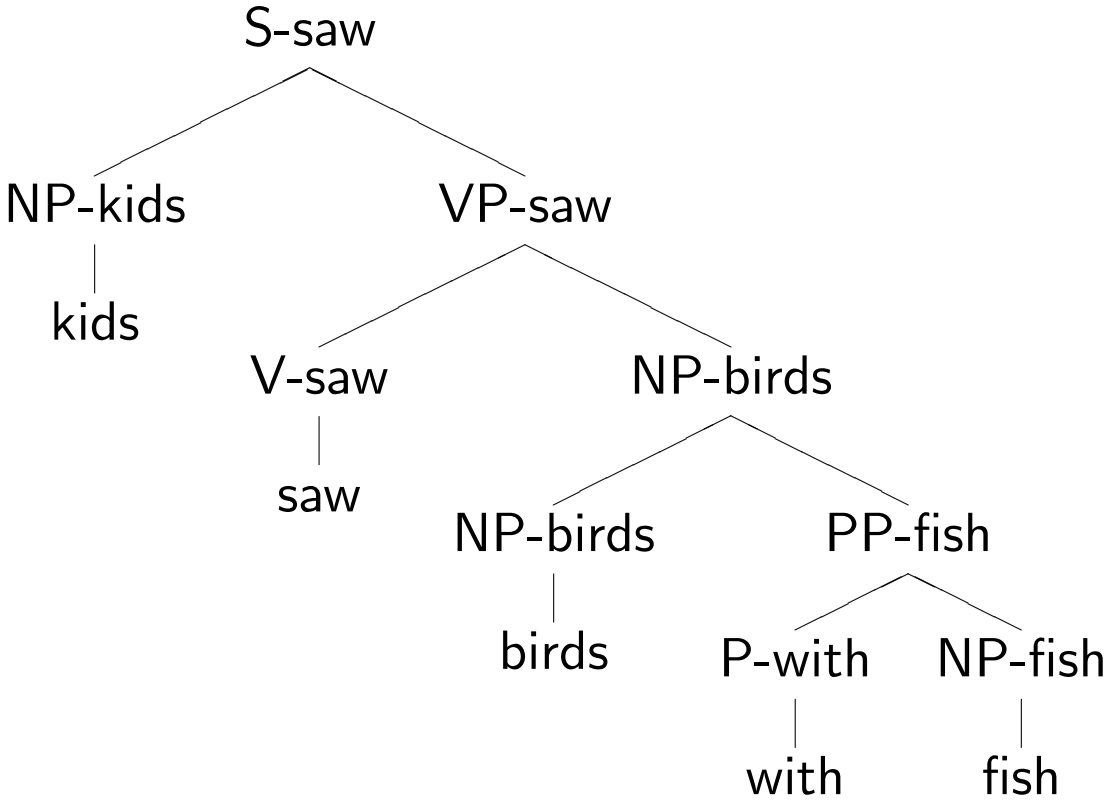
1. Dependencies: what/why

2. Transforming constituency $\rightarrow$ dependency parse

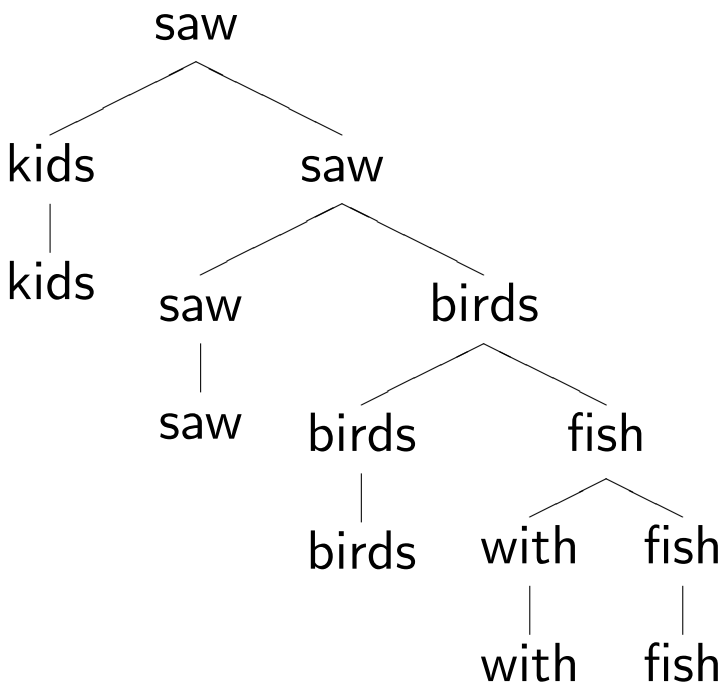
3. Direct dependency parsing

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

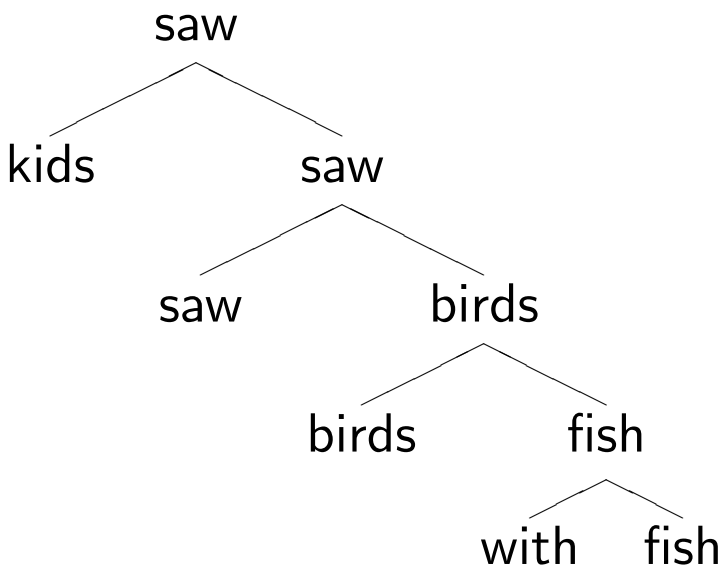
Lexicalized Constituency Parse



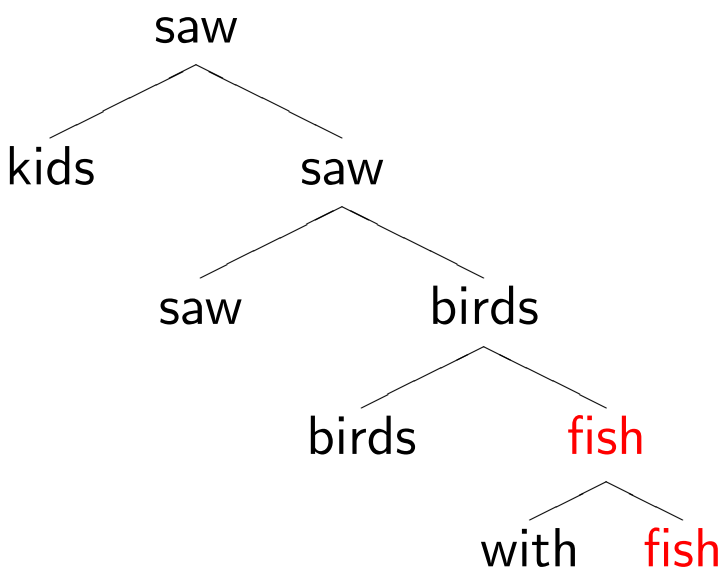
. . . remove the phrasal categories. . .



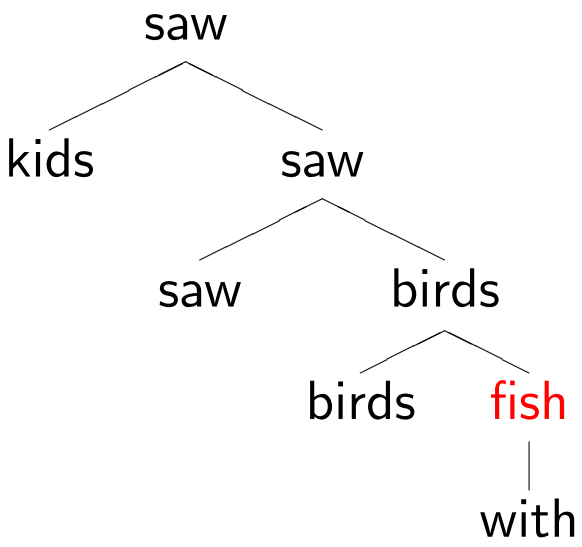
. . . remove the (duplicated) terminals. . .



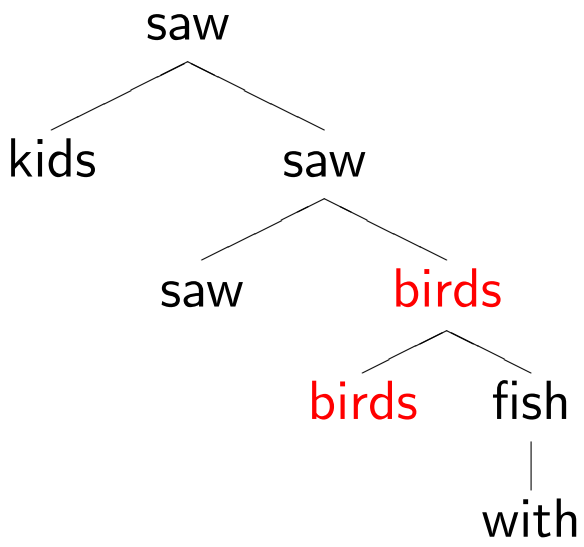
. . . and collapse chains of duplicates. . .



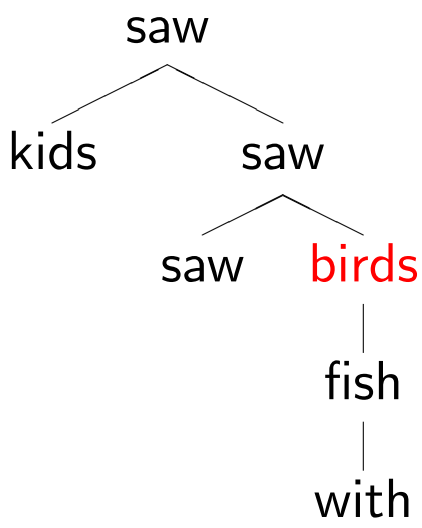
. . . and collapse chains of duplicates. . .



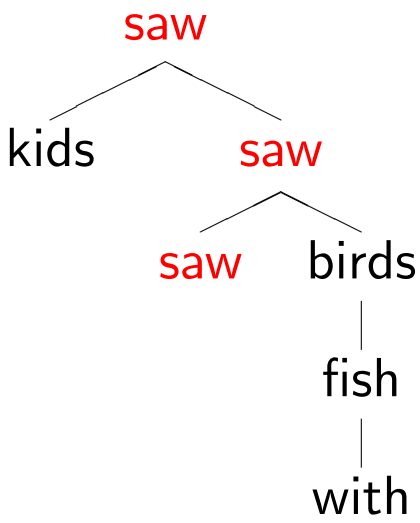
. . . and collapse chains of duplicates. . .



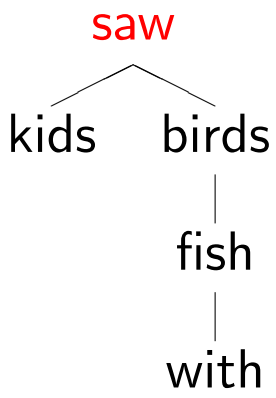
. . . and collapse chains of duplicates. . .



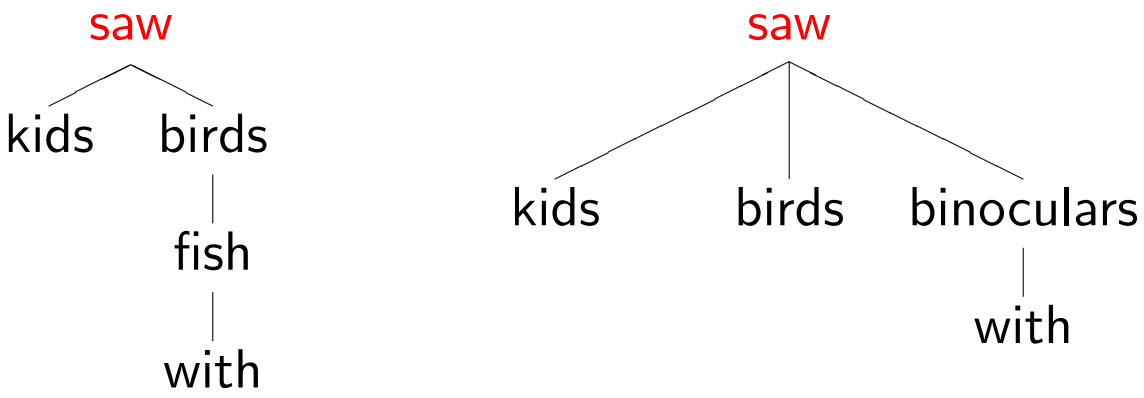
. . . and collapse chains of duplicates. . .



. . . and collapse chains of duplicates. . .



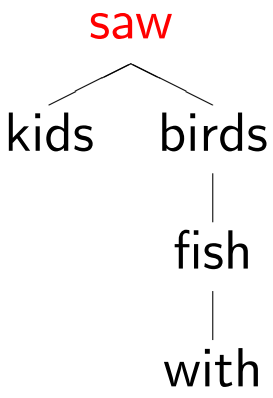
Dependency Parse

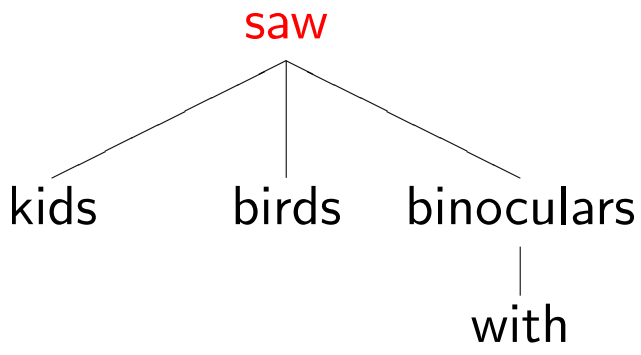


Linguists have long observed that the meanings of words within a sentence depend on one another, mostly in *asymmetric*, *binary* relations.

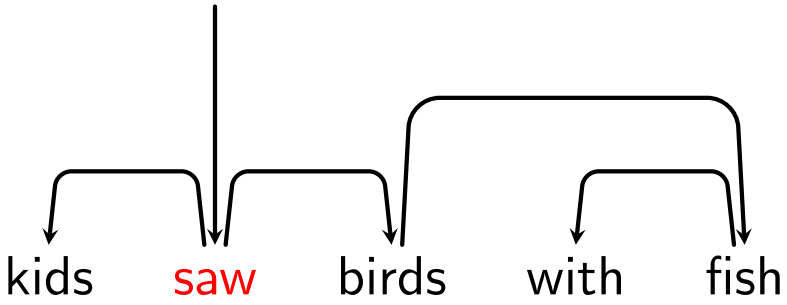
- Though some constructions don't cleanly fit this pattern: e.g., coordination, relative clauses.

Dependency Parse





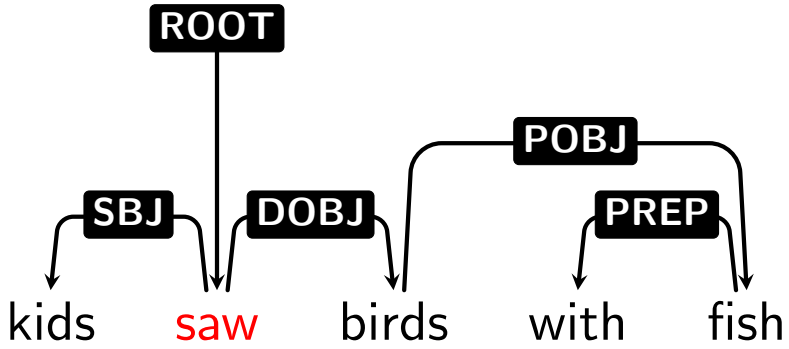
Equivalently, but showing word order (head → modifier):



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

Edge Labels

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

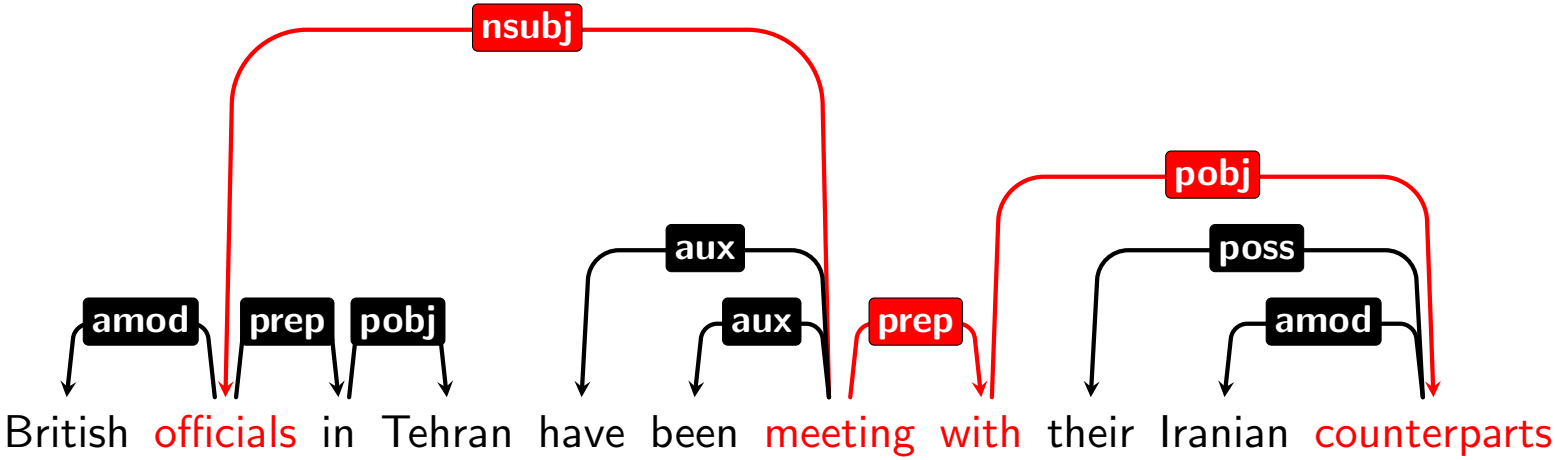


Important relations for English include subject, direct object, determiner, adjective modifier, adverbial modifier, etc. (Different treebanks use somewhat different label sets.)

- 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:

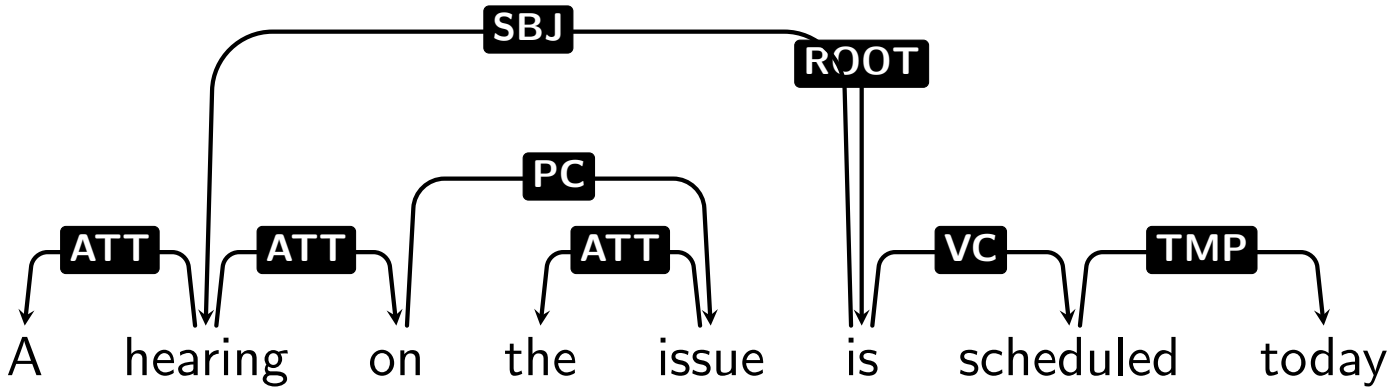


(example from Brendan O'Connor)

Projectivity

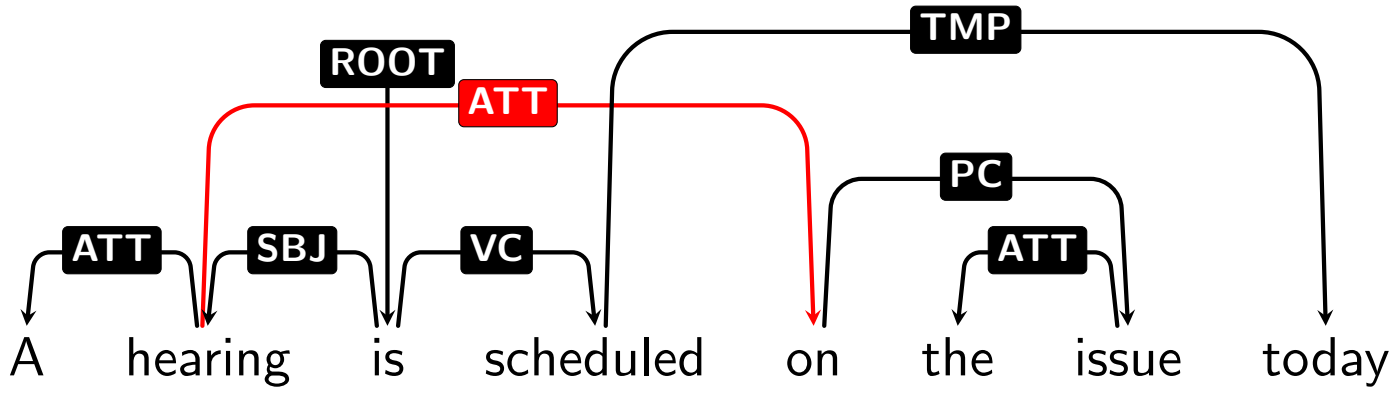
- A sentence's dependency parse is said to be **projective** if every subtree (node and all its descendants) occupies a *contiguous span* of the sentence.

- = 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

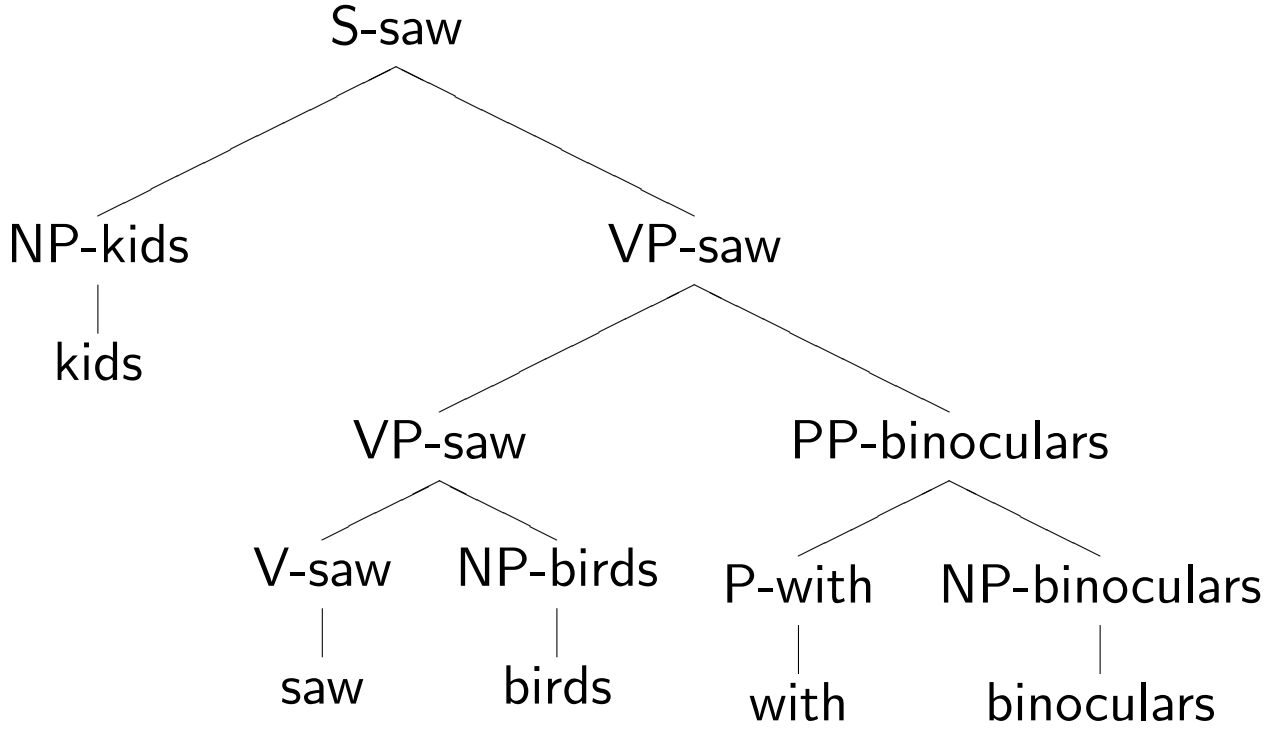
2. **Transforming constituency $\rightarrow$ dependency parse**

3. Direct dependency parsing

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

Constituency Tree → Dependency Tree

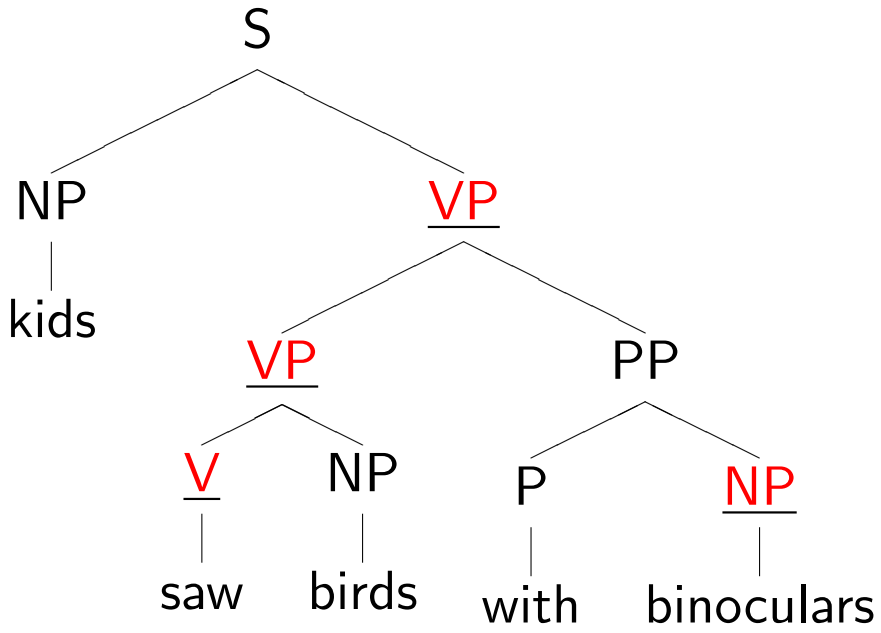
We saw how the **lexical head** of the phrase can be used to collapse down to a dependency tree:



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

Head Rules

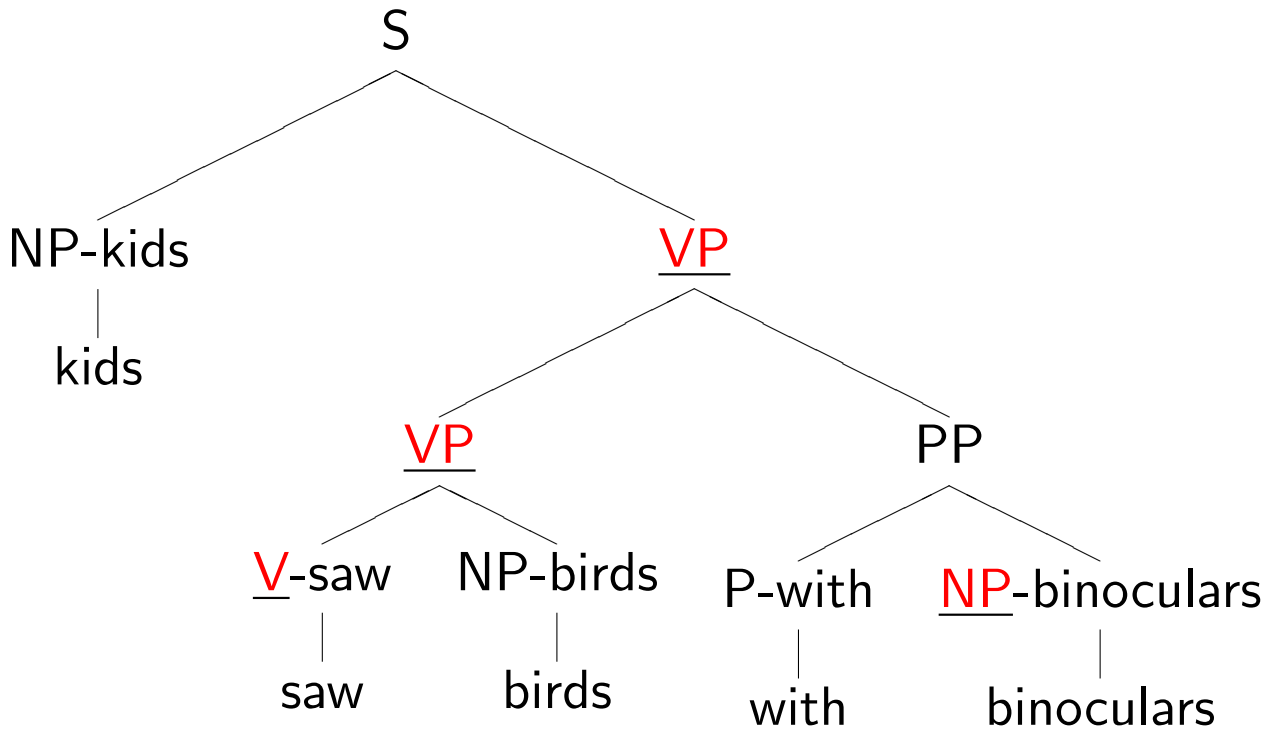
The standard solution is to use **head rules**: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. S → NP VP, VP → VP PP, PP → P NP (content head), etc.



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

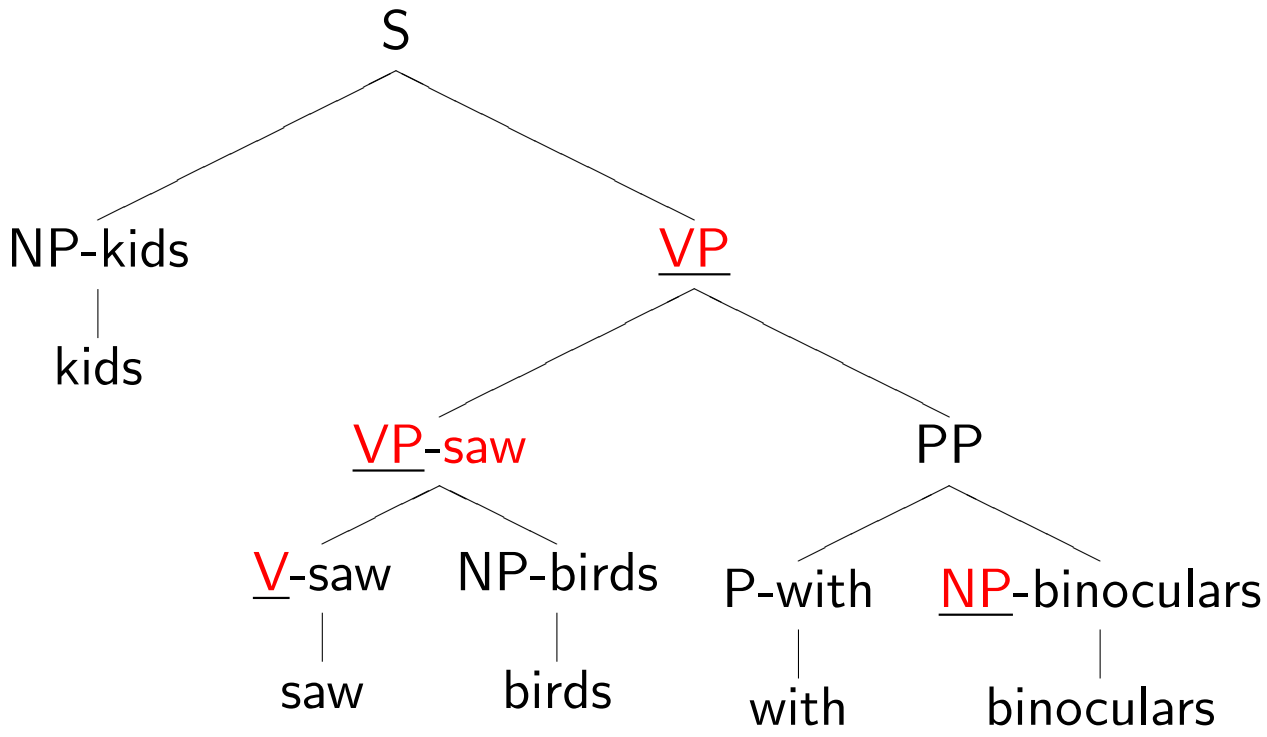
Head Rules

Then, propagate heads up the tree:



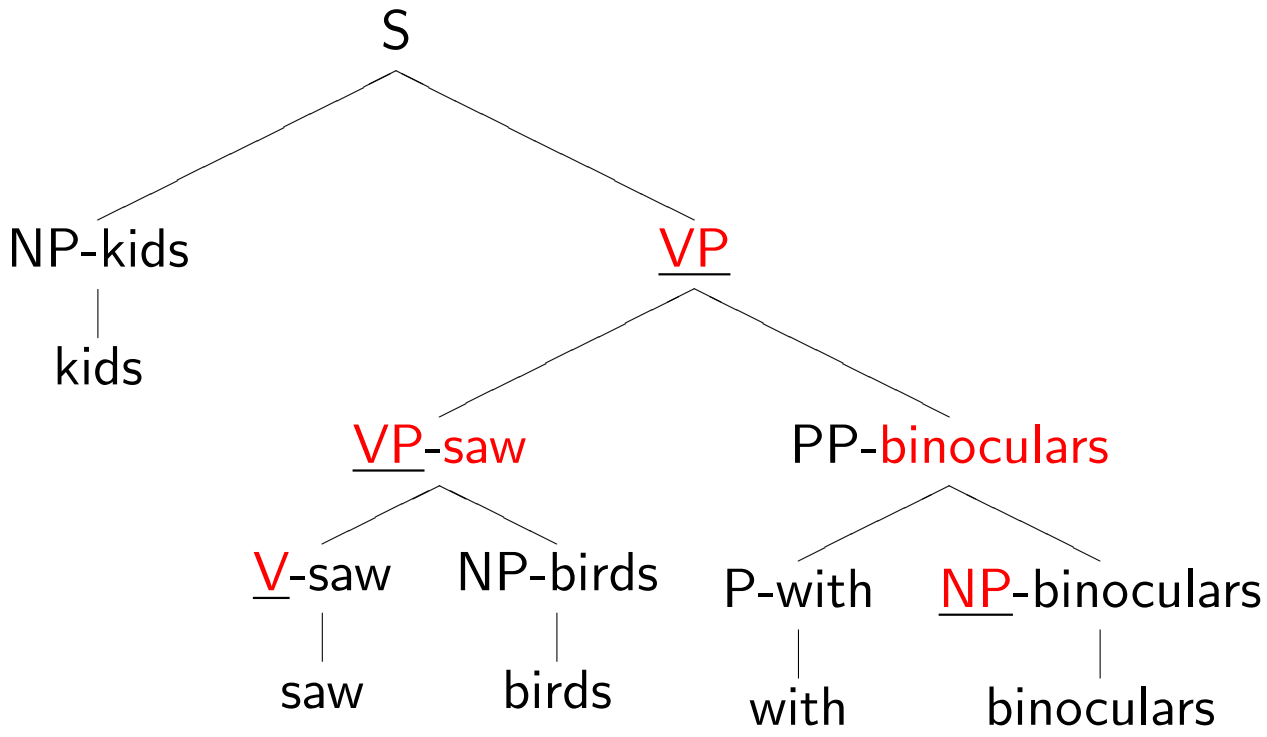
Head Rules

Then, propagate heads up the tree:



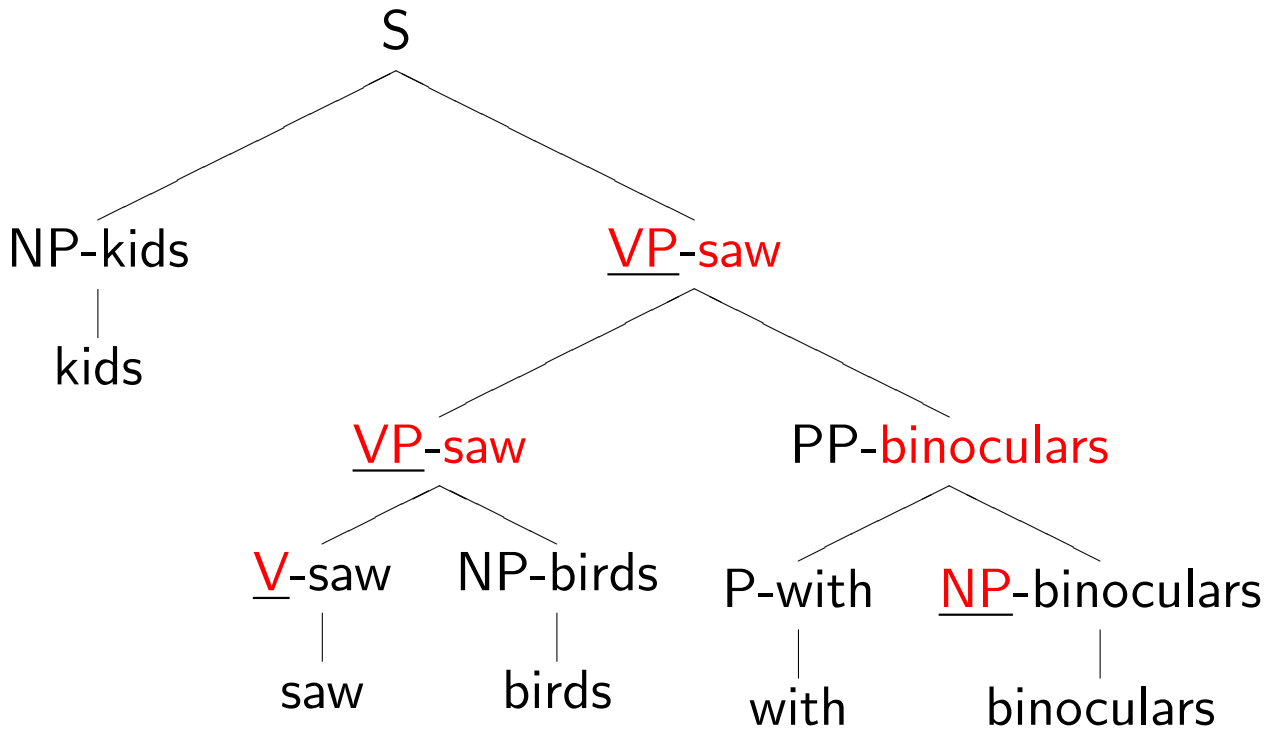
Head Rules

Then, propagate heads up the tree:



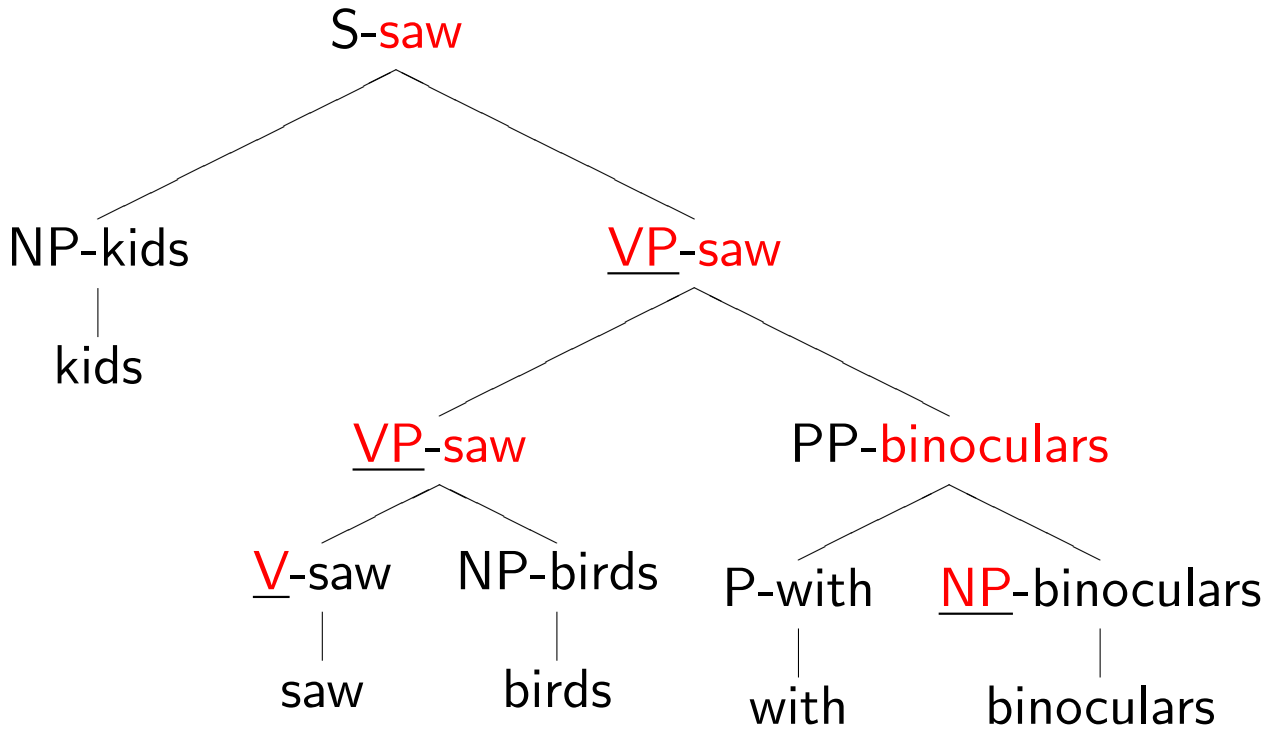
Head Rules

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**

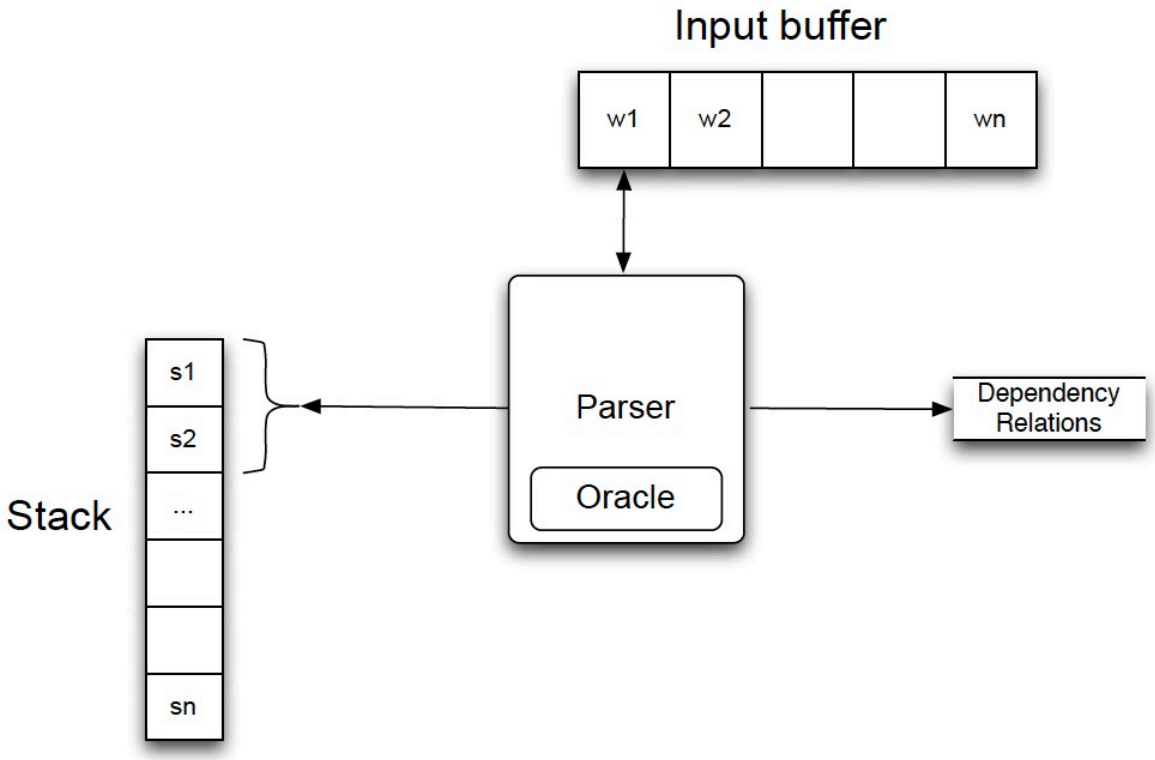
   - Transition-based (shift-reduce)
   - Graph-based

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.

Example

Step	Stack	Word List	Action	Relations
0	[root]	[Kim,saw,Sandy]		
1				
2				
3				
4				
5				

Kim saw Sandy

Example

Step	Stack	Word List	Action	Relations
0	[root]	[Kim,saw,Sandy]	Shift	
1	[root,Kim]	[saw,Sandy]		
2				
3				
4				
5				

Kim saw Sandy

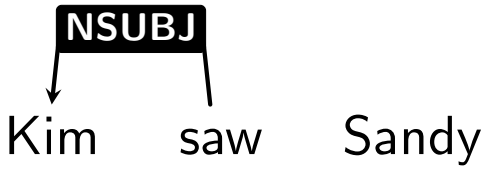
Example

Step	Stack	Word List	Action	Relations
0	[root]	[Kim,saw,Sandy]	Shift	
1	[root,Kim]	[saw,Sandy]	Shift	
2	[root,Kim,saw]	[Sandy]		
3				
4				
5				

Kim saw Sandy

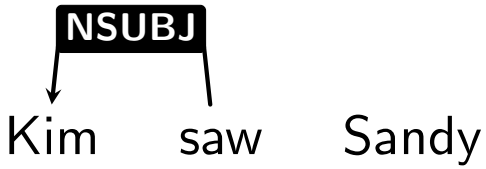
Example

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



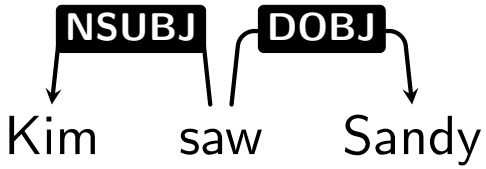
Example

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



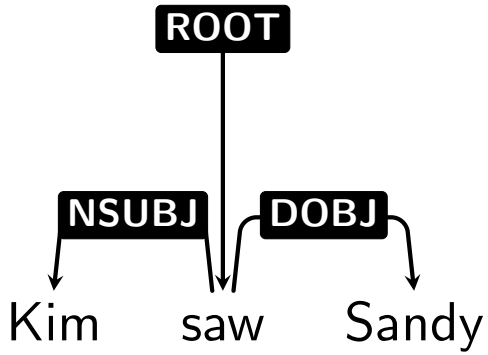
Example

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



Example

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



Transition-based Parsing

- Latent structure is just edges between words. Train a **classifier** 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, some treebanks are available solely in dependency form.

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.

- Two main paradigms, graph-based and transition-based, with different kinds of models and search algorithms.