```
Nom -> N
Nom -> Adj Nom
Nom -> Nom Conj Nom
```

S -> NP VP
NP -> D N (the cat)
NP -> PN N (my dog)
NP -> NP PP (paper about)
VP -> V NP
VP -> VP PP
VP -> VBD NP NP (give him a book)
PP -> P NP (at home)

Syntax & Parsing I

To capture long range dependencies

```
Following: constituency test (part of sentence is constituent or not)
```

To capture substitutability

POS categories indicate which words are substitutable (substituting adj)
Phrasal categories indicate which phrases are substitutable (substituting noun, constituency test)

Coordination

constituents coordinated using conj (insert "and")

clefting

Only a constituent can appear in the frame "____ is/are who/what/where/when/why/how …"

```
It is… that…
Wh- movement:
what is … ? […]
```

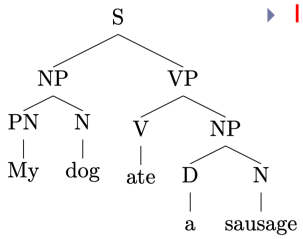
theory of syntax:

is sentence well formed? (!= meaningful)

Constituency (aka phrase) structures VS **Dependency** structures

different models of language behaviour (what can capture what, what parser)

Constituent trees definition:



Internal nodes (S, NP, VP, PP) = phrases (noun phrase, verb phrase, Prepositional phrase)

Nodes above words (PN, N, V) = PoS tagged (preterminals)

Context free grammar: (Non terminals (internal nodes + PoS tags), terminals, rules, start symbol)

Inner rules, Preterminal rules

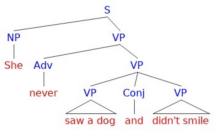
Context free: only affected by NT, not context

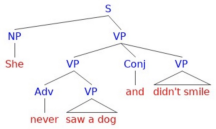
$$G = (V, \Sigma, R, S)$$

PoS ambiguity exampl

VBD		VB			
VBN	VBZ	VBP	VBZ		
NNP	NNS	NN	NNS	CD	NN

Fed raises interest rates 0.5 percent





Structural ambiguity (may caused by PoS Ambiguity)

PP **attachment ambiguity** not caused by PoS Ambi (e.g. I saw a girl with a telescope)

Conj attachment ambiguity: [Adj Nom Conj Nom]

Problems to solve:

Recognition problem: is there a tree yields the sentence?

Parsing problem: get most plausible tree

Parsing problem encompasses the recognition problem

Properties of parser:

Directionality: top down / bottom up (CYK) / mixed

Search strategy: DFS (need backtrack), BFS, best first

CYK:

Bottom up

supports only rules in the Chomsky Normal Form (CNF)

CNF):

$C \rightarrow x$ — Unary preterminal rules, generation of words given PoS tags $D \rightarrow the \quad N \rightarrow telescope$

$C \rightarrow C_1 C_2$ — Binary inner rules (e.g., $S \rightarrow NP\ VP,\ NP \rightarrow D\ \ N$)

CFG to CNF:

delete empty,

might delete unary rules

binarize trees (lossless Markovization in PCFG)

$NP \rightarrow DT\ \ NNP\ \ VBG\ \ NN$

$NP \rightarrow DT\ \ X$

$X \rightarrow NNP\ \ Y$

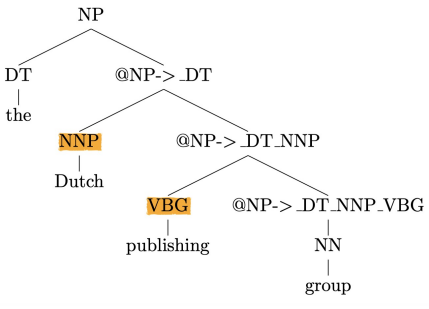
$Y \rightarrow VBG\ \ NN$

A more systematic way to refer to new non-terminals

$NP \rightarrow DT\ \ @NP|DT$

$@NP|DT \rightarrow NNP\ \ @NP|DT\_NNP$

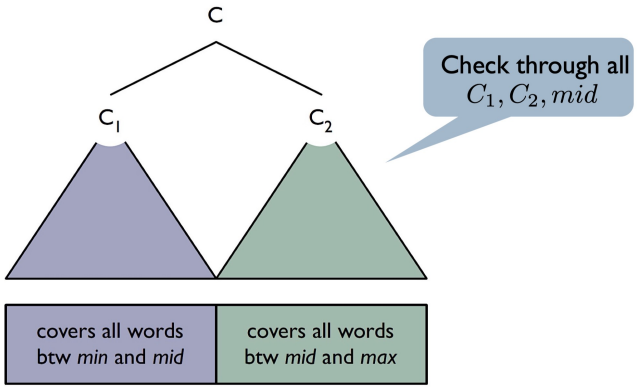
$@NP|DT\_NNP \rightarrow VBG\ \ NN$



Idea: dynamic programming

Parsing longer spans

$$C \rightarrow C_1 \ C_2$$



overgenerates: grammar generates ungrammatical noun phrases

e.g. the kids takes toys

mostly caused by not encoding number information (if something is singular or plural)

need to encode this information for determiners and nouns

make sure that only singular determiners and singular nouns can combine

Drawback: Sparse data (subcategorization)

might still generates noun phrases that are disfavoured if not ungrammatical

Syntax & Parsing II

Signature: for CYK

Applications of rules is independent of inner structure of a parse tree

We only need to know the corresponding span and the root label of the tree

- Its signature $[min, max, C]$
  span Rule label (numeric)

Intuition:

Compute for every span a set of admissible labels (may be empty for some spans)

- Start from small trees (single words) and proceed to larger ones

When done, check if S is among admissible labels for the whole sentence, if yes – the sentence belong to the language

- That is if a tree with signature *[0, n, S]* exists

e.g. (NP = S)

well-formed substring table (WFST):

NP	→	Det Nom	A	→	heavy | orange
Nom	→	book | orange | AP Nom	Det	→	my
AP	→	heavy | orange | Adv A	Adv	→	very

0 my 1 very 2 heavy 3 orange 4 book 5

from i to j

only i < j

upper triangle

from diagonal to top right

i \ j	1 my	2 very	3 heavy	4 orange	5 book
0 my	Det			NP → Det Nom	NP
1 very		Adv	AP → Adv A	Nom	Nom
2 heavy			A,AP	Nom → AP Nom	Nom
3 orange				Nom,A,AP	Nom
4 book					Nom

Formal Rep:

Chart can be represented by a Boolean array `chart[min][max][C]`

- Relevant entries have $0 \leq$ `min` < `max` $\leq$ `n`

`chart[min][max][C]` = `true` if the signature (min, max, C) is already added to the chart; `false` otherwise.

the	women	fish	with	bait
1: DT	10: NP(1,2) 2: N, 3: NP(2)	11: NP(2,5) 4: V, 5: N, 6: NP(5)	7: P	15: S(10,13) 14: S(3,13) 13: VP(4,12) 12: PP(7,9) 8: N, 9: NP(8)

Number the steps, write the combined steps to back track

Rule to look to left & bottom:
Could use [bottom count] + 1 spaces
e.g. 15 could use (4+1) spaces, thus left 3, bottom 2

Implementations…

Implementation: binary rules

```
for each max from 2 to n
  for each min from max - 2 down to 0
    for each syntactic category C
      for each binary rule C -> C1 C2
        for each mid from min + 1 to max - 1
          if chart[min][mid][C1] and chart[mid][max][C2] then
            chart[min][max][C] = true
```

= in the graph, combine each col value with inverse of row value, vice versa

Skeleton

```
// int n = number of words in the sequence
// int m = number of syntactic categories in the grammar
// int s = the (number of the) grammar's start symbol
boolean[][][] chart = new boolean[n + 1][n + 1][m]
// Recognize all parse trees built with with preterminal rules.
// Recognize all parse trees built with inner rules.
return chart[0][n][s]
```

Solve Recognition problem: Is there a tree?

Tackle Unary: extend the grammar, to get the transitive closure

$$\begin{matrix} A \to B \\ B \to C \end{matrix} \quad \Rightarrow \quad \begin{matrix} A \to B \\ B \to C \\ A \to C \end{matrix}$$

Time complexity

$\theta(n^3|R|)$, where $|R|$ is the number of rules in the grammar

n: one sentence length

probabilistic CFG

deal with ambiguity (solve parsing problem)
score (how plausible) all the derivations (trees)
P(T) = Multiply all probabilities

Associate probabilities with the rules $p(X \to \alpha)$:

$$\forall\, X \to \alpha \in R: \quad 0 \le p(X \to \alpha) \le 1$$

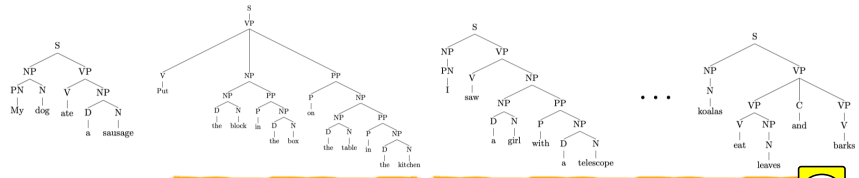
$$\forall X \in N: \quad \sum_{\alpha: X \to \alpha \in R} p(X \to \alpha) = 1$$

All [emit from] sums to 1

e.g. all LHS sums to 1

$VP \to V$	0.2
$VP \to V\ NP$	0.4
$VP \to VP\ PP$	0.4
$N \to girl$	0.2
$N \to telescope$	0.7
$N \to sandwich$	0.1

A **treebank:** a collection sentences annotated with constituent trees



An estimated **probability of a rule** (**maximum likelihood estimate**)

$$p(X \to \alpha) = \frac{C(X \to \alpha)}{C(X)}$$

The number of times the rule used in the corpus

The number of times the nonterminal X appears in the treebank

e.g.

Rule	Count	Prob. estimate
$S \to B\ C$	n_1	$n_1/(n_1+n_2+n_3)$
$S \to C$	n_2	$n_2/(n_1+n_2+n_3)$
$S \to B$	n_3	$n_3/(n_1+n_2+n_3)$

Smoothing is helpful

- Especially important for **preterminal rules**, i.e. generation of words (= the task model in PoS tagging)
- The same smoothing techniques as studied before can be used (e.g., **GT or add 1 smoothing**)

parsing with PCFGs:

Solve Parsing Problem: **get most probable tree**

Goal: define a distribution over parse trees

Let us denote by $G(x)$ the **set of derivations** (Trees) for the **sentence** x

The **probability distribution** defines the **scoring** $P(T)$ over the **trees** $T \in G(x)$

Finding the **best parse** for the sentence according to PCFG:

$$\arg\max_{T \in G(x)} P(T)$$

Unfortunately, not all PCFGs give rise to a proper distribution over trees, i.e. the **sum over probabilities** of all trees the **grammar can generate may be less than 1**: $\sum_T P(T) < 1$

= prob of all PoS ambiguous trees for A SENTENCE might < 1

any PCFG estimated with the **maximum likelihood** procedure are **always proper**

CKY: (double array instead of boolean)

- Chart is represented by a double array `chart[min][max][C]`
  - It stores probabilities for the most probable subtree with a **given signature**
- `chart[0][n][S]` will store the **probability** of the **most probable full parse tree**

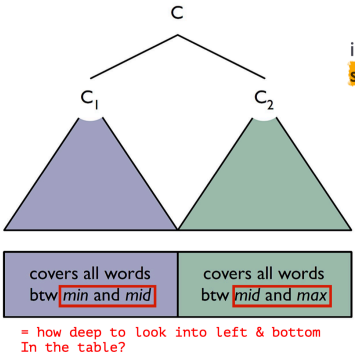
Intuition:

$$C \rightarrow C_1 \ C_2$$

For every C choose C_1, C_2 and mid such that

$$P(T_1) \times P(T_2) \times P(C \rightarrow C_1 C_2)$$

is maximal, where T_1 and T_2 are left and right subtrees.



Implementation:

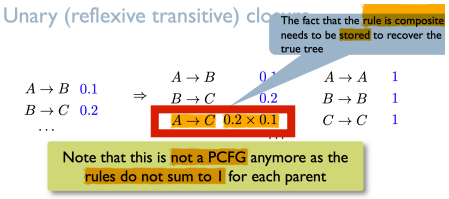
preterminal rules

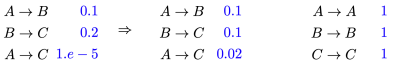
```
for each wi from left to right
          word
  for each preterminal rule C -> wi

    chart[i - 1][i][C] = p(C -> wi)
```

Binary rules:

```
for each max from 2 to n
  for each min from max - 2 down to 0
    for each syntactic category C
      double best = undefined
      for each binary rule C -> C1 C2
        for each mid from min + 1 to max - 1
          double t1 = chart[min][mid][C1]
          double t2 = chart[mid][max][C2]
          double candidate = t1 * t2 * p(C -> C1 C2)
          if candidate > best then
            best = candidate
      chart[min][max][C] = best
```

Unary (reflexive transitive) closure





Recovery of the tree

- For each signature we store backpointers to the elements from which it was built (e.g., rule and, for binary rules, midpoint)
  - start recovering from *[0, n, S]*
- Be careful with unary rules
  - Basically you can assume that you always used an unary rule from the closure (but it could be the trivial one $C \rightarrow C$)

Speeding up

Basic pruning (roughly):

- For every span (i,j) store only labels which have the probability at most N times smaller than the probability of the most probable label for this span
- Check not all rules but only rules yielding subtree labels having non-zero probability

Coarse-to-fine pruning

- Parse with a smaller (simpler) grammar, and precompute (posterior) probabilities for each spans, and use only the ones with non-negligible probability from the previous grammar

Syntax & Parsing III: Constituent Parsing

Intrinsic evaluation:

Exact match: percentage of trees predicted correctly

Bracket score: scores how well individual phrases (and their boundaries) are identified

Crossing brackets: percentage of phrases boundaries crossing

Dependency metrics: scores dependency structure corresponding to the constituent tree (percentage of correctly identified heads)

Bracket score:

Tree = collection of brackets: [min, max, C]

Compare bracket against gold standard

Use Precision, recall and F1

Bracket example:

$$Pr = \frac{\text{number of brackets the parser and annotation agree on}}{\text{number of brackets predicted by the parser}}$$

$$Re = \frac{\text{number of brackets the parser and annotation agree on}}{\text{number of brackets in annotation}}$$

$$F1 = \frac{2 \times Pr \times Re}{Pr + Re}$$

Harmonic mean of precision

(S
(NP (PN My) (N Dog))
(VP (V ate)
(NP (D a) (N sausage))

- Pre-terminals (lexical categories) don't count as constituents (but unit rules do)

(treebank) PCFGs (Directly read-off rules from the treebank) **drawbacks**:

not encode lexical preferences

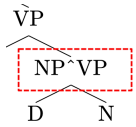
not encode structural properties

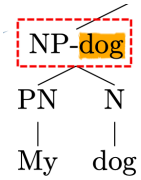
e.g. Emission of NPs under S vs. NPs under VP are very different

Independence assumptions in PCFGs are too strong for this grammar

Following approaches: Increase F1 score

Solution: Structural annotation, Lexicalization





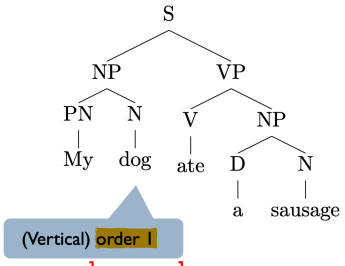
Which one?
Follow head rule

S -> NP VP
NP -> D N (the cat)
NP -> PN N (my dog)
NP -> NP PP (paper about)
VP -> V NP
VP -> VP PP
PP -> P NP (at home)

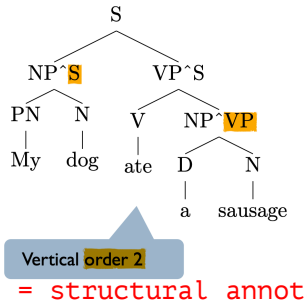
transforming trees, then inducing a PCFG, instead of transforming the grammar

In treebank

Grammar for parsing unchanged



= unchanged



= structural annotation

Only annotate internal nodes,
Not preterminals

Vertical Markovization (similar to 2nd order HMM)

Rule applications depend on past ancestors (similar to Structural annotation)

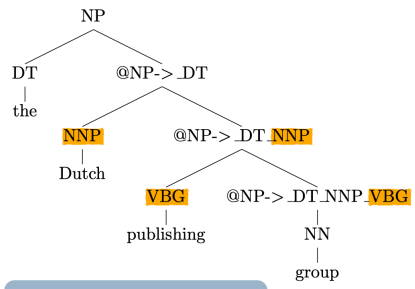
Close attachment (PCFG weakness): = PP attachment ambiguity

same rules are used in 2 different trees, PCFG return the same scores

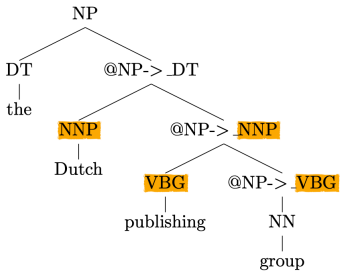
solved by vertical Markovization: enrich PCFG, assign higher probability to some rule, break close attachment

Horizontal Markovization (horizontal history) = systematic way to transform to CNF

In vertical Markovization we increased context, in horizontal Markovization we want to reduce it



= original way to transform to CNF
Encode all history,
Child = parent || neighbor`



Only encode the root (NP) as parent

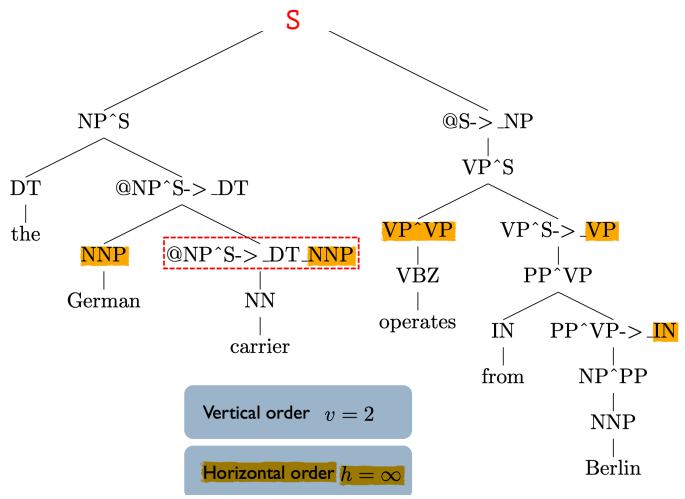
A more systematic way to refer to new non-terminals

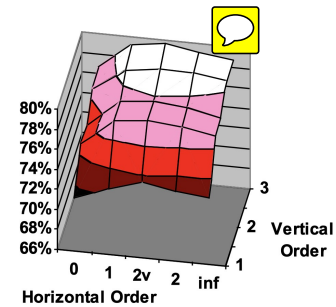
$NP \rightarrow DT$ @NP|DT

@NP|DT → NNP @NP|DT_NNP

@NP|DT_NNP → VBG NN

Both Vertical & Horizontal Markovization, Increase F1





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

Splitting: PoS tag

Enrich grammar, subcategorization

Also increase F1 score

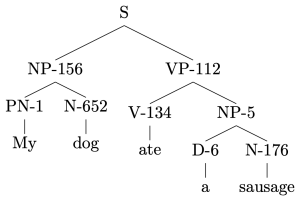
- PoS tags in Penn Treebank are too coarse
- Very obvious for IN tag:
  - Assigned both to 'normal' prepositions (to form a prepositional phrase) – *in, on, at*, … –
  - and to subordinating conjunctions (e.g., *if*)
    - *E.g., check if advertising works*

- 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
= still context free

inducing splits (through EM): learning NTs from data, automatically enrich grammar

= clustering of tree contexts of non-terminal symbols



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:

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

Use NN to predict

Lecture 13 - Heads and Dependency Parsing

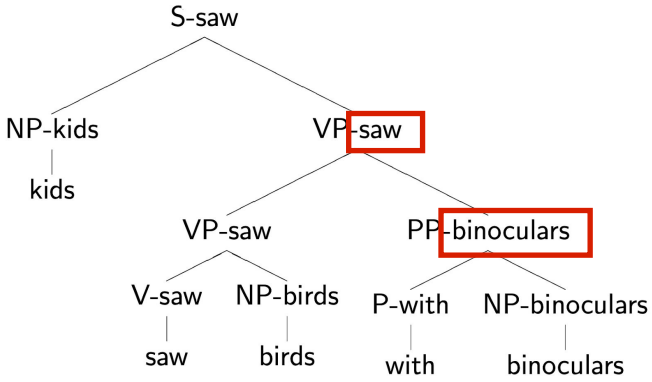
PCFGs have No lexical dependencies (since context free)

= Same parsing if replacing words

This approach fix PCFG without using constituent structure

Handling Lexical Dependencies (Lexicalisation)

- 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*
- **Lexicalisation:** create new categories by adding **lexical head** of the phrase
  - Example:



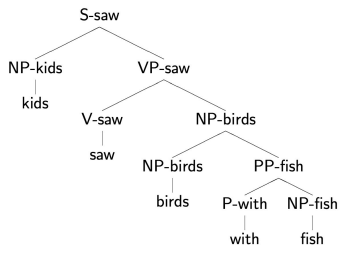
  - Now consider: VP - saw -> VP - saw PP - fish vs. VP - saw -> VP - saw PP - binoculars
- Practical issues
  - Identifying the head of every rule is not always straightforward
  - All this category-splitting makes the grammar much more **specific** (good!)
    - But leads to huge grammar blowup and very **sparse** data (bad!)

Solution: Do we *really* need phrase structure in the first place? Not always!

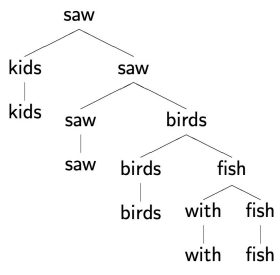
Dependency Trees

constituency → dependency

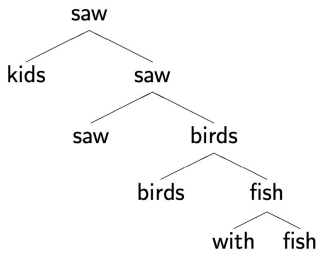
1. Original tree



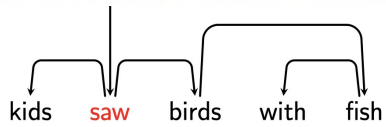
2. Remove phrasal categories



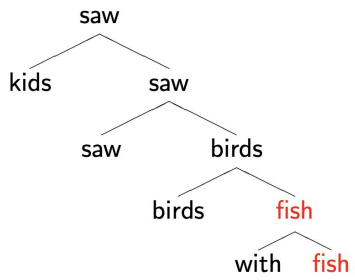
3. Remove duplicated terminals



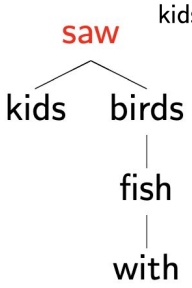
word order (head → modifier):



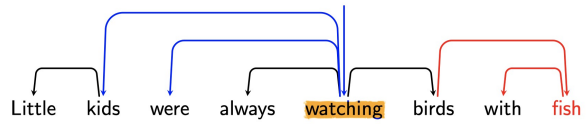
4. Collapse chains of duplicates



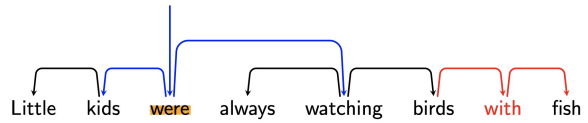
5. Result



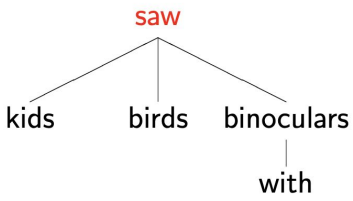
Some treebanks prefer **content heads**:



Others prefer **functional heads**:



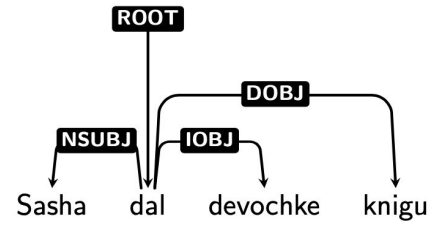
- Don't worry if the tree looks stupid to you, because it is!
  - This demonstrates how dependency trees can help with correct parsing too
  - The right one is

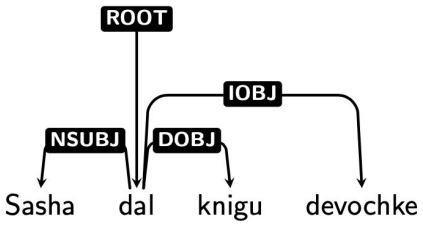


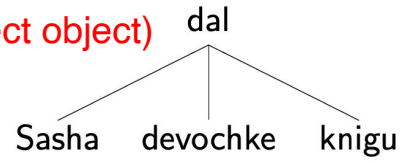
- Meaning of words within a sentence depend on one another, mostly in **asymmetric**, **binary** relations
  - Though some constructions don't clearly fit this pattern: e.g. coordination and relative clauses
- Also, in languages with **free word order** (e.g. Turkish, Russian), phrase structure (constituency) grammars don't make as much sense
  - E.g. we would need both $S \rightarrow NP\ VP$ and $S \rightarrow VP\ NP$
    - Not very informative about what is really going on
    - In contrast, dependency relations stay constant

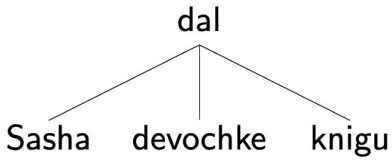
LABELS:

Root
Det: the
nSubj: subject
dobj: direct object
iobj: indirect object
(receives the direct object)
nmod: where?
amod: adj
Case: prep (in, from)



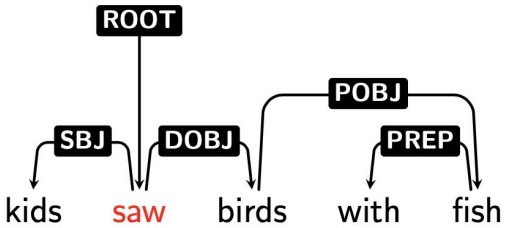






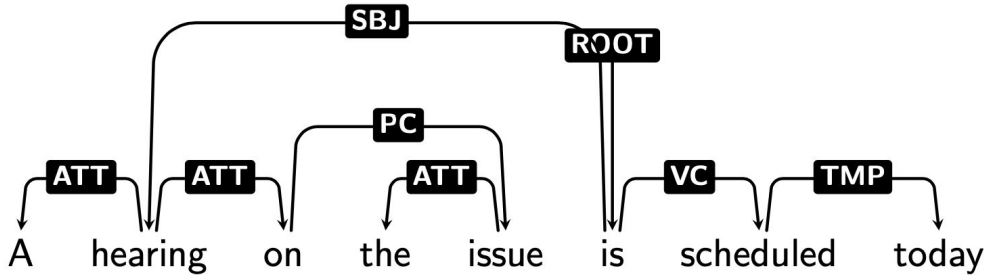
- Edge labels can help us distinguish between different kinds of $head \rightarrow modifier$ relations:

Provide good feature for info extraction task



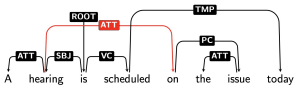
  - Important relations for English include *subject, direct object, determiner, adjective, modifier, adverbial modifier, etc.*

- 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



  - In which case, *the dependency parse can be drawn on top of the sentence without any crossing edges*
  - Non-projectivity is rare in English but quite common in many other languages

= crossing edges



Head Rules

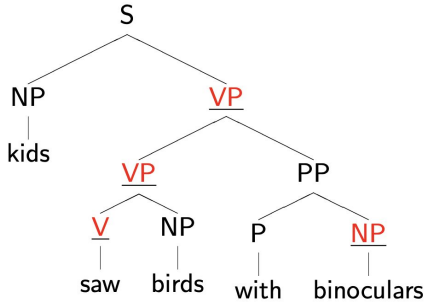
- How can we find each phrase's head in the first place?
- The standard solution is to use **head rules**

Right hand side

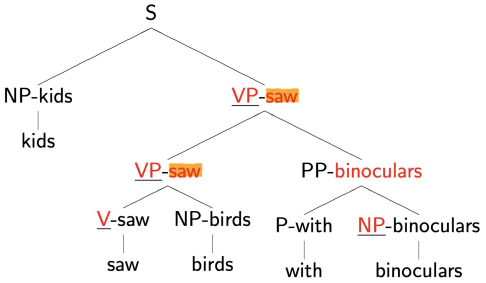
  - For every non-unary (P)CFG production, designate on RHS non-terminal as containing the head
    - E.g. *S -> NP VP*

Head & Parse Rules:

S -> NP VP
NP -> D N (the cat)
NP -> PN N (my dog)
NP -> NP PP (paper about)
VP -> V NP
VP -> VP PP
PP -> P NP (at home)



Then, propagate heads up the tree:



  - We can also employ heuristics to scale this to large grammars: *within an NP, last immediate N child is the head*

Pros and Cons

- Pros
  - 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, information-extractors, etc.
  - Parsers can be very fast
- Cons

= arrows from word1 to word2

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

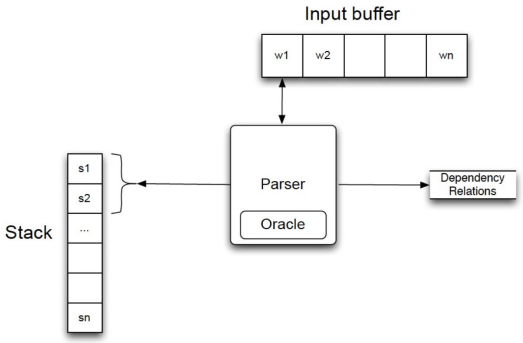
Direct Dependency Parsing

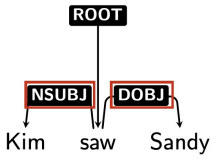
- **CKY** can be adapted, though efficiency is a concern: obvious approach is $O(Gn^5)$; Eisner algorithm brings it back down to $O(Gn^3)$
- **Shift-reduce** is more efficient, and doesn't even require a grammar!

Shift-Reduce Parsing

Notice 2 ways of parsing gives same result:
e.g. the cat chased a dog

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] Pop S2	[Sandy]	Shift	
4	[root,saw,Sandy]	[]	RightArc	dobj(saw,Sandy)
5	[root,saw] Pop S1	[]	RightArc	root→saw
6	[root]	[]		





- 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 relations point from head to dependent**
- Both **LeftArc** and **RightArc** leave the head at the top of the stack

Transition-Based Parsing

- Latent structure is just edges between words
- Train a **classifier** as the oracle to predict next action (Shift, LeftArc, RightArc), and proceed left-to-right through the sentence
- $O(n)$ time complexity!
- Only finds **projective** trees (without special extensions)

Graph-Based Parsing

= connect every pair of words, then prune

- From a fully connected directed graph of all possible edges, choose the best ones that form a tree
- **Edge-factored** models:
  - Classifier assigns a non-negative score to each possible edge
  - **Maximum spanning tree** algorithm finds the spanning tree with the highest total score in $O(n^2)$ time
- Can be formulated as constraint-satisfaction problem with **integer linear programming**

Compariso

- Transition-Based shift reduce + classifier
  - Scoring function can look at any part of the stack
  - No optimality guarantees for search
  - Linear-time
  - (Classically) Projective only
- Graph-Based
  - Scoring function limited by factorisation
  - Optimal search within the model
  - Quadratic time
  - No projectivity constraint
- Conversion-Based = CYK
  - In terms of accuracy, *sometimes* best to first constituency-parse, then convert to dependencies (e.g. Stanford Parser)
  - Slower than direct methods and need 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.
- Two main paradigms, graph-based and transition-based, with different kinds of models and search algorithms.