Computational Linguistics

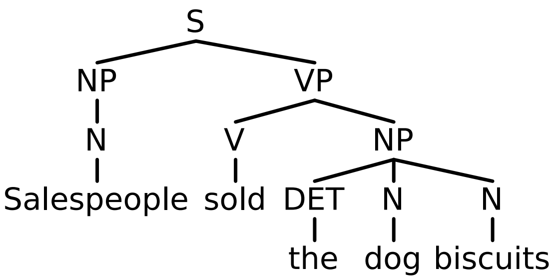
Probabilistic Parsing

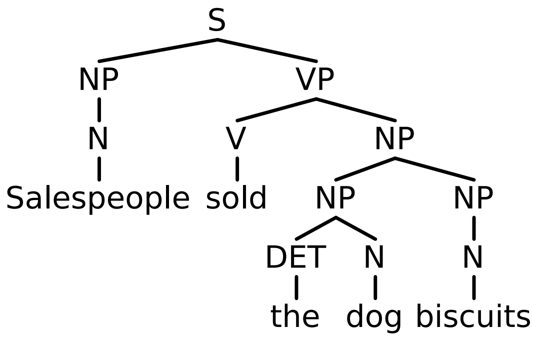
Dietrich Klakow & Stefan Thater
FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes

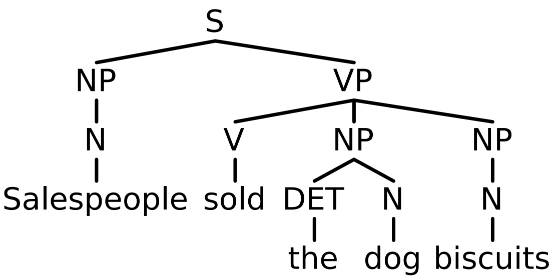
Summer 2012

(Charniak, 1997)

Salespeople sold the dog biscuits







S → NP VP	NP → NP NP
VP → V NP	NP → N
VP → V NP NP	DET → the
NP → DET N	N → dog
NP → DET N N	…

Ambiguity & Disambiguation

- **Probabilistic disambiguation**
  choose the one that is most derivation tree if the input sentence is ambiguous (has > 1 derivation trees)
- **We need …**
  - a probabilistic model of (contex-free) grammar
  - methods to estimate probabilities

Further Motivation

- **Natural language is ambiguous**
  ⇒ disambiguation
- **Grammar development**
  ⇒ automatically induce grammars
- **Efficient search**
  ⇒ compute the most likely parse tree first
- **Robustness**

Probabilistic Context-Free Grammars (PCFG)

- **Probabilistic context-free grammar (PCFG)**
  - a context-free grammar ⟨V, Σ, R, S⟩
  - a funktion P assigning a value p ∈ [0, 1] to each rule
    - such that $\sum_{\beta \in V^*} P(A \to \beta) = 1$
- $P(A \to \beta)$ = the conditional probability that symbol A is expanded to β
  - Alternative notations: $P(\beta \mid A)$, $P(A \to \beta \mid A)$, $A \to \beta$ [p]

Derivation Trees (Recap)

- Derivarion trees:
  - The root node is labeled with the start symbol S
  - Leaf nodes are labeled with terminal symbols or ε
  - An inner node and their child nodes correspond to the rules that have been used in the derivation
- **Parsing:**
  Compute all derivation trees for a given input
- **Probabilistic parsing:**
  Compute the most likely derivation tree

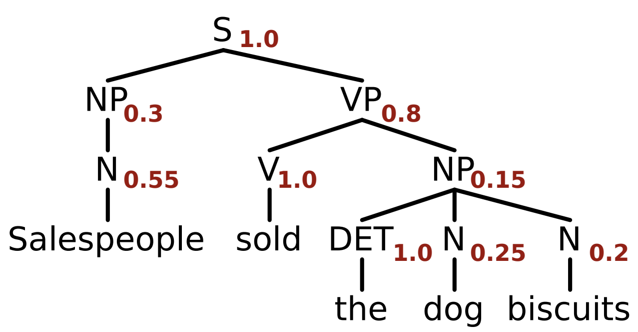
Probabilistic Context-Free Grammar (PCFG)

- A PCFG assigns a probability to each derivation tree of a sentence.
- **The probability of a derivation tree** T is defined as the product of the probabilities of all the rules that have been used to expand the nodes in T:
  - **P(T, w)** = $P(T) = \prod_{n \in T} P(R(n))$
  - R(n) is the rule that has been used to expand node n
  - Note: $P(T, w) = P(T)\ P(w \mid T) = P(T)$, because $P(w \mid T) = 1$
- **The probability of a sentence** w is the sum of the probabilities of all its derivation trees:
  - $P(w) = \Sigma_T\ P(w, T)$, for $w \in L(G)$

(Charniak, 1997)

Salespeople sold the dog biscuits

S → NP VP	[1.0]
VP → V NP	[0.8]
VP → V NP NP	[0.2]
NP → DET N	[0.5]
NP → N	[0.3]
NP → DET N N	[0.15]
NP → NP NP	[0.05]
DET → *the*	[1.0]
N → *Salespeople*	[0.55]
N → *dog*	[0.25]
N → *biscuits*	[0.2]
V → *sold*	[1.0]

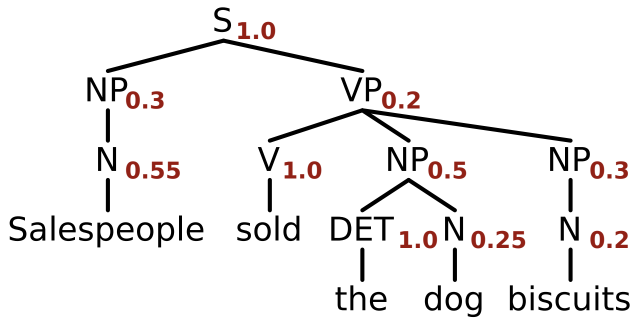


$$P(t) = 1.0 \times 0.3 \times 0.55 \times 0.8 \times 1.0 \times 0.15 \times 1.0 \times 0.25 \times 0.2 = 9.9 \times 10^{-4}$$

(Charniak, 1997)

Salespeople sold the dog biscuits

S → NP VP	[1.0]
VP → V NP	[0.8]
VP → V NP NP	[0.2]
NP → DET N	[0.5]
NP → N	[0.3]
NP → DET N N	[0.15]
NP → NP NP	[0.05]
DET → *the*	[1.0]
N → *Salespeople*	[0.55]
N → *dog*	[0.25]
N → *biscuits*	[0.2]
V → *sold*	[1.0]

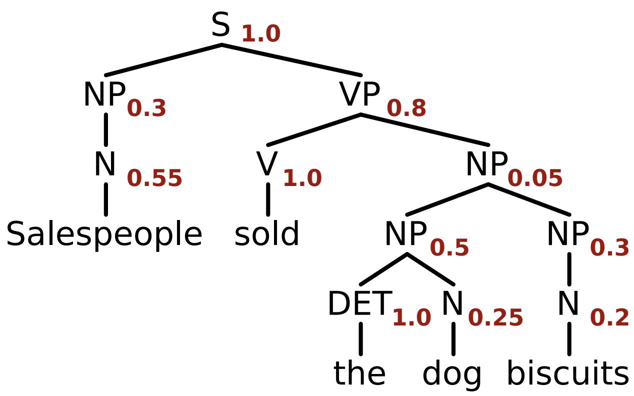


$$P(t) = 1.0 \times 0.3 \times 0.55 \times 0.2 \times 1.0 \times 0.5 \times 1.0 \times 0.25 \times 0.3 \times 0.2 = 2.475 \times 10^{-4}$$

(Charniak, 1997)

Salespeople sold the dog biscuits

S → NP VP	[1.0]
VP → V NP	[0.8]
VP → V NP NP	[0.2]
NP → DET N	[0.5]
NP → N	[0.3]
NP → DET N N	[0.15]
NP → NP NP	[0.05]
DET → *the*	[1.0]
N → *Salespeople*	[0.55]
N → *dog*	[0.25]
N → *biscuits*	[0.2]
V → *sold*	[1.0]



$$P(t) = 1.0 \times 0.3 \times 0.55 \times 0.8 \times 1.0 \times 0.05 \times 0.5 \times 1.0 \times 0.25 \times 0.3 \times 0.2 = 4.95 \times 10^{-5}$$

Probabilistic Context-Free Grammar (PCFG)

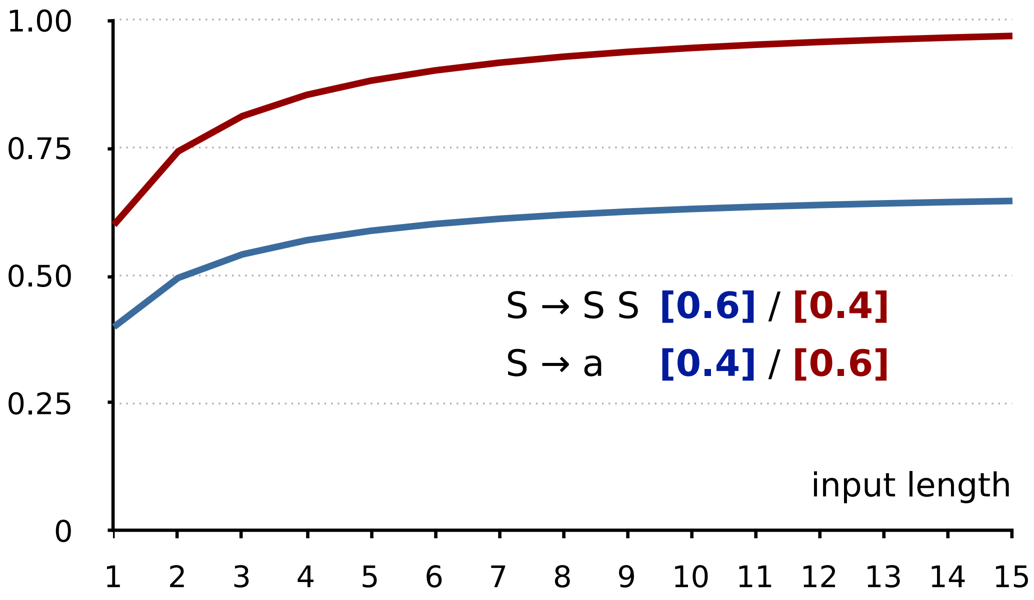
- The probability of a sentence w is the sum of the probabilities of all its derivation trees:
  - $P(w) = \Sigma_T P(w, T)$, for $w \in L(G)$
- A PCFG G is **consistent** if $\Sigma_{w \in L(G)} P(w) = 1$
- Recursion can lead to inconsistent grammars:
  - S → S S [0.6]
  - S → a [0.4]

An inconsistent PCFG

- S → S S [0.6] / [0.4]
- S → a [0.6] / [0.6]

- $P(a^i) = \#trees(a^i) \times 0.6^{i-1} \times 0.4^i = 0.4$
  - P(a) = 0.4, P(aa) = 0.096, P(aaa) = 0.0461, …
- $P(a^i) = \#trees(a^i) \times 0.4^{i-1} \times 0.6^i = 0.4$
  - P(a) = 0.6, P(aa) = 0.144, P(aaa) = 0.06912, …
- Number of trees (#trees) for a^{i+1} = *i*-th Catalan number

An inconsistent PCFG

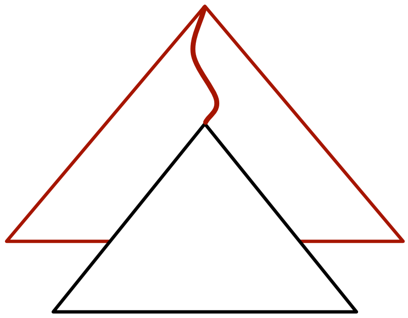


Probabilistic Parsing

- **Language modelling ("inside probabilities")**
  compute the probability that S ⇒* w for an input sentence w:
  - $P(w) = \sum_T P(w, T)$
- **Probabilistic parsing ("viterbi scores")**
  compute the most likely derivation tree T(w) for an input sentence w:
  - $$T(w) = \arg\max_T P(T \mid w) = \arg\max_T \frac{P(T, w)}{P(w)} = \arg\max_T P(T)$$

Properties of PCFGs

- The probability of a (sub) tree is indipendant of
  - the context in which the tree occurs
  - the node(s) that dominates the tree



Probabilistic CYK Parsing

- Extend the CYK algorithm:
  - T[i, j, A] = the probability that A $\Rightarrow^*$ $w_{i+1} \dots w_j$
- **Inside probabilities:**
  - T[i, j, A] = sum of the probabilities of all derivation trees of the substring $w_{i+1} \dots w_j$
- **Probability of a derivation tree (parsing)**
  - T[i, j, A] = the probability of the most likely derivation
  - B[i, j, A] = the corresponding derivation tree

CYK (without probabilities)

```
function CYK(G, w₁ ... wₙ):
   for i in 1 ... n do
      T[i-1, i] = { A | A → wᵢ ∈ R }
      for j in i - 2 ... 0 do
         T[j, i] = ∅
         for k in j + 1 ... i - 1 do
            T[j, i] = T[j, i] ∪
               { A | A → B C, B ∈ T[j,k], C ∈ T[k, i] }
         done
      done
   done
   if S ∈ T[0, n] then return True else return False
```


CYK (with probabilities)

```
function CYK(G, w₁ ... wₙ):
   ⟨initialize T and B⟩
   for i in 1 ... n do
      for all nonterminals A in G do
         T[i-1, i, A] = P(A → wᵢ)
      for j in i - 2 ... 0 do
         for k in j + 1 ... i - 1 do
            for all A → B C do
               pr = T[j, k, B] × T[k, i, C] × P(A → B C)
               if pr > T[j, i, A] then
                  T[j, i, A] = pr
                  B[j, i, A] = ⟨construct subtree⟩
   return ⟨B[0, n, S] and T[0, n, S]⟩
```

Learning PCFG Probabilities

- **Option #1**
  count frequencies of rules in syntactically annotated treebanks (such as the Penn Treebank)
- **Option #2**
  Inside-outside algorithm (not discussed here)

Learning PCFG Probabilities

- We are given a syntactically annotated corpus
  - annotated corpus = a set of derivation trees
- We can construct a grammar from the treebank by identifying the rules with all “subtrees” of height 1
- **Estimating rule probabilities:**
  - $P(A \rightarrow \alpha) = \dfrac{\text{count}(A \rightarrow \alpha)}{\sum_{\beta} \text{count}(A \rightarrow \beta)}$
  - count(A → α) = the number of times the rule A → α has been used in all trees in the corpus

(Example: Webber/Keller)

Learning PCFG Probabilities

- **A very small treebank:**
  - S_1: [$_S$ [$_{NP}$ grass] [$_{VP}$ grows]]
  - S_2: [$_S$ [$_{NP}$ grass] [$_{VP}$ grows] [$_{AP}$ fast]]
  - S_3: [$_S$ [$_{NP}$ grass] [$_{VP}$ grows] [$_{AP}$ slowly]]
  - S_4: [$_S$ [$_{NP}$ bananas] [$_{VP}$ grow]]
- **Rules & rule probabilities:**
  - S → NP VP 2/4
  - S → NP VP AP 2/4
  - NP → grass 3/4
  - …

Learning PCFG Probabilities

	Rule	P(A → α)
r_1	S → NP VP	2/4
r_2	S → NP VP AP	2/4
r_3	NP → grass	3/4
r_4	NP → bananas	1/4
r_5	VP → grows	3/4
r_6	VP → grow	1/4
r_7	AP → fast	1/2
r_8	AP → slowly	1/2

Learning PCFG Probabilities

- **Probabilities of the sentences:**
  - $P(S_1) = P(r_1) \times P(r_3) \times P(r_5) = 2/4 \times 3/4 \times 3/4 = 0.28125$
  - $P(S_2) = P(r_2) \times P(r_3) \times P(r_5) \times P(r_7) = 0.140625$
  - $P(S_3) = P(r_2) \times P(r_3) \times P(r_5) \times P(r_7) = 0.140625$
  - $P(S_4) = P(r_1) \times P(r_4) \times P(r_6) = 0.03125$

Evaluation

- **Coverage:** How many sentences are well-formed according to the grammar?
- **Accuracy:** How many sentences are correctly parsed?
  - measured as “relative correctness” wrt. to category label, start and end position (yield) of all constituents (subtrees)
  - **Labelled precision:** percentage of correct subtrees in the parser output
  - **Labelled recall:** percentage of correct subtrees in the gold standard (test corpus)

Evaluation

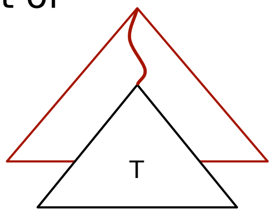
- **Labelled Precision** = C / M
- **Labelled Recall** = C / N
- where
  - C = number of correct constituents produced by the parser
  - M = total number of constituents produced by the parser
  - N = total number of constituents in reference corpus

Binarization

- Replace rules of the form $A \rightarrow A_1\ A_2\ A_3\ \ldots\ A_k$ [p] by
  - $A \rightarrow \langle A_1,\ldots,A_{k-1}\rangle\ A_k$ [p]
  - $\langle A_1,\ldots,A_{k-1}\rangle \rightarrow A_1\ \ldots\ A_{k-1}$ [1.0]
- … or binarize trees in the treebank before “reading off” the grammar from the trees.

Problems

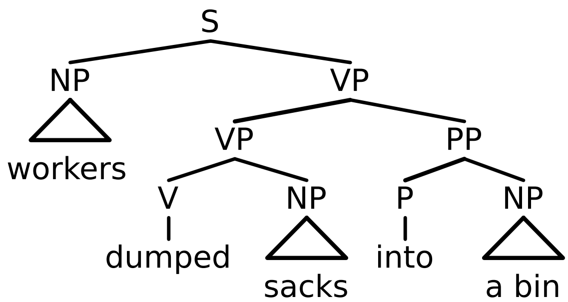
- The probability of a (sub) tree is indipendant of
  - the context in which the tree occurs
  - the node(s) that dominates the tree

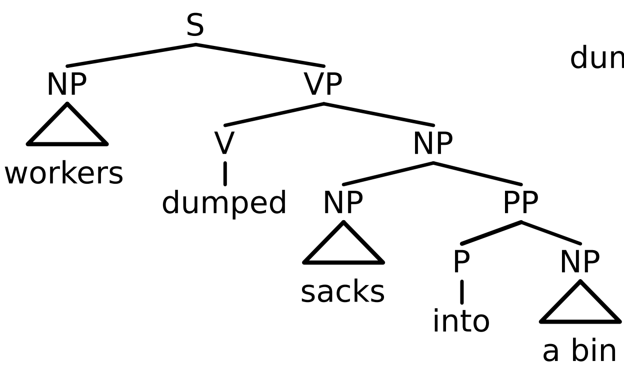


- **Problems:** we *want* to capture …
  - Lexical dependencies
  - Structural dependencies

Lexical Dependencies

- The two trees differ only in one rule:
  - VP → VP PP
  - NP → NP PP





Lexical Dependencies

- The two trees differ only in one rule:
  - VP → VP PP
  - NP → NP PP
- ⇒ the grammar will either
  - always prefer the 1st rule (VP attachment) or
  - always prefer the 2nd rule (NP-attachment)
- **But …**
  - *Workers dumped sacks into a bin*
  - *Fishermen caught tons of herring*
- ⇒ Lexikalized PCFG

(Manning & Schütze)

Lexical Dependencies

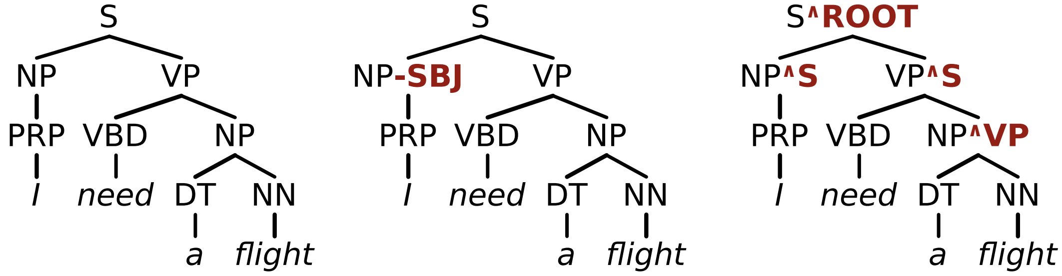
	come	**take**	**think**	**want**
VP → V	*9.5%*	*2.6%*	*4.6%*	*5.7%*
VP → V NP	*1.1%*	*32.1%*	*0.2%*	*13.9%*
VP → V PP	*35.5%*	*3.1%*	*7.1%*	*0.3%*
VP → V SBAR	*6.6%*	*0.3%*	*73.0%*	*0.2%*
VP → V S	*2.2%*	*1.3%*	*4.8%*	*70.8%*
VP → V NP S	*0.1%*	*5.7%*	*0.0%*	*0.3%*
VP → V PRT NP	*0.3%*	*5.8%*	*0.0%*	*0.0%*
VP → V PRT PP	*6.1%*	*1.5%*	*0.2%*	*0.0%*
…	…	…	…	…

Structural dependencies

- **Structural independencies:**
  - The (probability of an) application of a rule is independent of all other rules in the derivation tree
  - NP → Pronoun vs. NP → Det Noun
    same probabilities for all occurrences of NP
- **But …** (Francis &al, 1999)
  - Subject-NP: 91% pronouns, 9% non-pronouns
  - Object-NP: 34% pronouns, 66% non-pronouns
  - (Switchboard corpus, spoken language)
- ⇒ Parent annotation

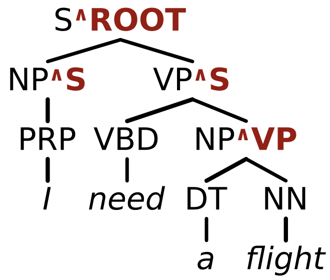
Structural dependencies

- Some dependencies can be “built into” the category symbols.



Structural dependencies

- **Parent Annotation:** nodes are annotated with the label of their parent nodes
- Similar effect compared to conditional probabilities
  - P(NP^S → PRP)
  - P(NP → PRP | S)



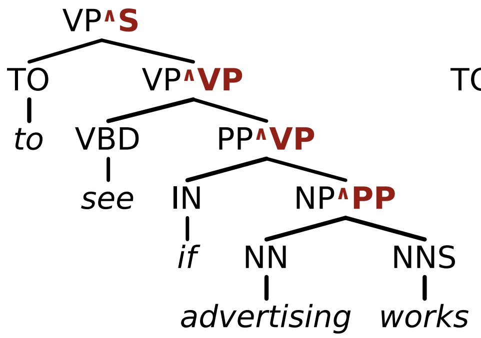
- Compare:
  - P(NP-SBJ → PRP) – no correspondence to conditional probabilities

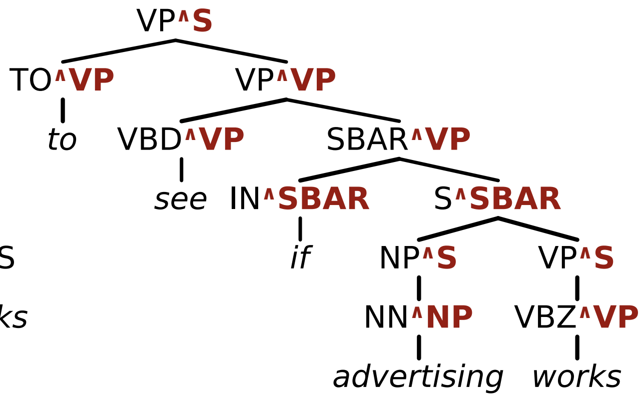
Structural dependencies

- Parent annotation can also be useful for preterminal nodes
- Most frequent adverbs with parent …
  - ADVP – *also, now*
  - VP – *not, n’t*
  - NP – *only, just*
- Penn Treebank – no distinction (same POS) between
  - subordinating conjunctions (*while, as, if*),
  - complementizers (*that, for*)
  - prepositions (*of, in, from*)

Structural dependencies

- Parent annotation can also be useful for preterminal nodes





Structural dependencies

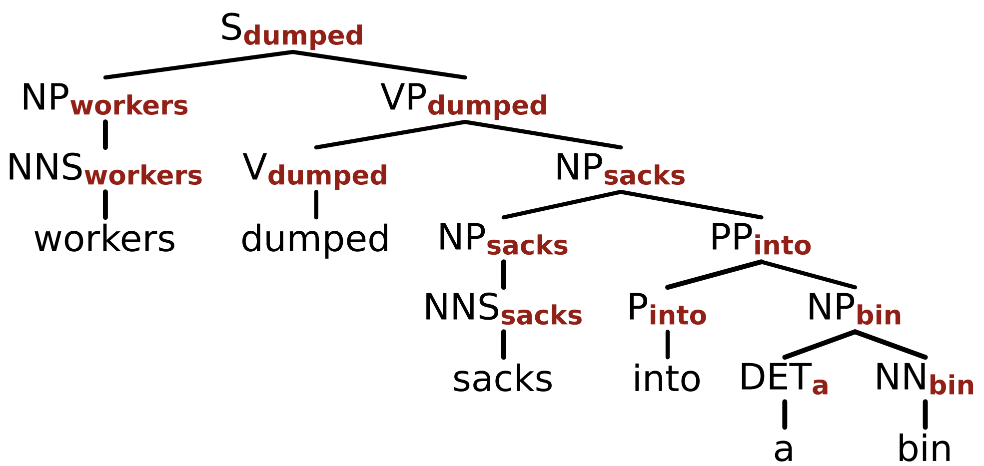
- **Parent annotation – drawbacks**
  - the grammar gets larger
  - fewer training data for each rule
  - reduced generalization (“overfitting”)

Lexical dependencies

- **The head** of a constituent is the “central” word of a phrase
  - Noun – NP
  - Verb – VP, S
  - Adjektive – AP
  - Preposition – PP

Lexical dependencies

- **Lexicalized parsing:** annotate nodes with their lexical heads



Lexical dependencies

	Rule	$P(A \rightarrow \alpha)$
r_1	$S_{dumpled} \rightarrow NP_{workers}\ VP_{dumped}$	1/1
r_2	$NP_{workers} \rightarrow NNS_{workers}$	1/1
r_3	$NP_{sacks} \rightarrow NNS_{sacks}$	1/2
r_4	$NP_{sacks} \rightarrow NP_{sacks}\ PP_{into}$	1/2
r_5	$NP_{bin} \rightarrow DT_{a}\ NN_{bin}$	1/1
…	…	…

Lexical dependencies

- **Problems:**
  - this leads to much larger grammars
  - its hard to estimate the rule probabilities

Lexicalized parsing

- **Complexity (CYK)**
  - Runtime: $O(|rules|n^3)$,
  - Wost case: $|rules| = |nonterminals|^3$
- **Lexicalized grammars**
  - Worst case: $|rules| = |nonterminals|^3 \cdot |terminals|^2$
  - |terminals| usually much larger than |nonterminals|
  - $\Rightarrow O(n^5)$ runtime for typical grammars and input sentences

Literature

- Jurafsky & Martin (2009) Speech and Language Processing Kapitel 14.
- Manning & Schütze (1999). Foundations of Statistical Natural Language Processing. Kapitel 11 & 12.
- Eugene Charniak (1993). Statistical Language Learning. Kapitel 5.