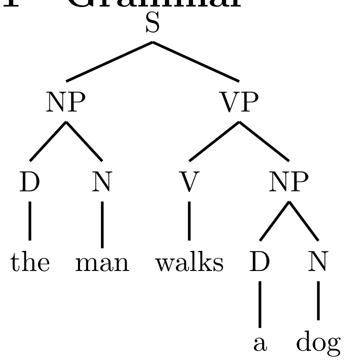
# NLP: Parsing

AMIT DHOMNE

## 1 Grammar




- "Consitiuency" parse
- S (sentence), NP (noun phrase), VP (verb phrase) are constituents
- Words combine to make phrases, and phrases combine to make larger phrases and sentences.

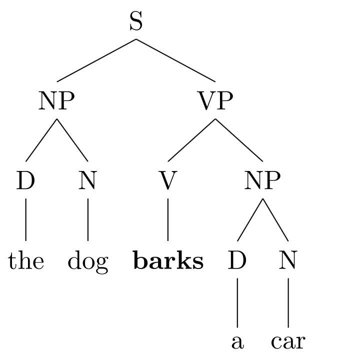
## 2 Context-Free Grammars (CFG)

- Context-free grammars can be specified by a table of "productions"

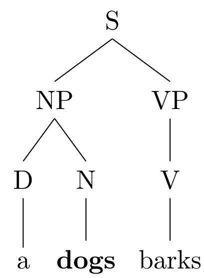
| | | | | | | |
|---|---|---|---|---|---|---|
| S | $\rightarrow$ | NP VP | | D | $\rightarrow$ | {*the, a*} |
| NP | $\rightarrow$ | D N | ("a dog barks") | N | $\rightarrow$ | {*man, dog*} |
| NP | $\rightarrow$ | N | ("dogs bark") | | | |
| VP | $\rightarrow$ | V NP | (transitive verb) | V | $\rightarrow$ | {*barks, walks, saw*} |
| VP | $\rightarrow$ | V | (intransitive verb) | | | |

- Words are called "terminals" and other nodes are "non-terminals"

- Independence assumption: each rule choice is dependent only on the parent node; it is independent of all other rule choices
- Independence assumption leads to many problems. A couple:



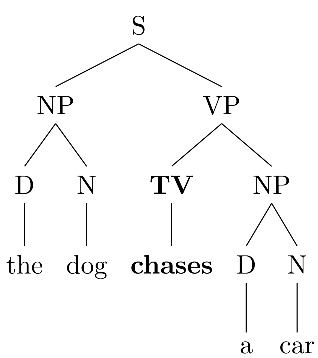
(a) Intransitive verb with a direct object



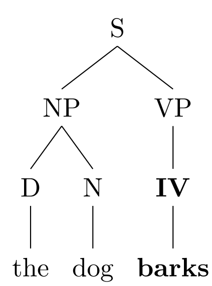
(b) Noun doesn't agree in number with determiner or verb

- We can solve some of these issues by refining our CFG. For example:

| | | | | | | |
|---|---|---|---|---|---|---|
| VP | $\rightarrow$ | IV | (intransitive verb) | IV | $\rightarrow$ | {*barks, walks*} |
| VP | $\rightarrow$ | TV NP | (transitive verb) | TV | $\rightarrow$ | {*chases, walks*} |



(c) Intransitive verbs are disallowed



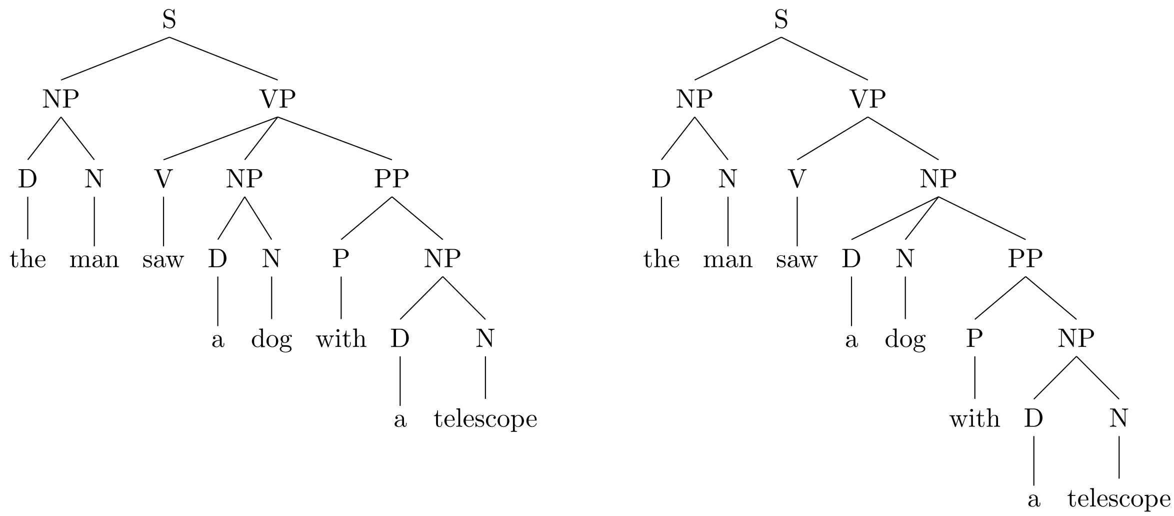
(d) Transitive verbs are disallowed

## 3 Syntactic Ambiguity

- For a given CFG, there can be multiple trees that describe the same sentence
- Add the following rules to the above:

| | | | | | | |
|---|---|---|---|---|---|---|
| NP | $\rightarrow$ | D N PP | (prep phrase attach to noun) | N | $\rightarrow$ | $\{$*telescope*$\}$ |
| VP | $\rightarrow$ | V NP PP | (prep phrase attach to verb) | | | |
| PP | $\rightarrow$ | P NP | ("in the house") | | | |

- "The man saw a dog with a telescope"




- Ambiguity is explosive
- A sentence ending in $n$ prepositional phrases has *over* $2^n$ parses (catalan numbers)
  - "I saw the man with the telescope": 2 parses
  - "I saw the man on the hill with the telescope": 5 parses
  - "I saw the man on the hill in texas with the telescope": 14 parses
  - "I saw the man on the hill in texas with the telescope at noon": 42 parses
  - "I saw the mon on the hill in texas with the telescope at noon on monday": 132 parses

## 4 Agreement

- In order for a sentence to be grammatical, we must also respect *agreement rules*
  - number: *a dog* vs. *all dogs*
  - person: 1st person (*I am*), 2nd person (*you are*), 3rd person (*he is*)
  - gender: *un homme* vs. *use femme* or even *she sees herself*

- The grammar defined above does not enforce this
  - For an NP, since productions are independent of each other, we could get either *dog* or *dogs* no matter whether the D is *a* or *all*
- We can incorporate this into our grammar by duplicating our productions to ensure agreement:

| | | | |
|---|---|---|---|
| S | $\rightarrow$ | $NP_{sg}$ $VP_{sg}$ | |
| S | $\rightarrow$ | $NP_{pl}$ $VP_{pl}$ | |
| $NP_{pl}$ | $\rightarrow$ | $N_{pl}$ | ("$\underline{\text{dogs}}$ bark") |
| $NP_{pl}$ | $\rightarrow$ | $D_{pl}$ $N_{pl}$ | ("$\underline{\text{all dogs}}$ bark") |
| $NP_{sg}$ | $\rightarrow$ | $D_{sg}$ $N_{sg}$ | ("$\underline{\text{a dog}}$ barks") |
| $VP_{sg}$ | $\rightarrow$ | $V_{sg}$ | ("a dog $\underline{\text{barks}}$") |
| $VP_{sg}$ | $\rightarrow$ | $V_{sg}$ $NP_{sg}$ | ("a dog $\underline{\text{chases a car}}$") |
| $VP_{sg}$ | $\rightarrow$ | $V_{sg}$ $NP_{pl}$ | ("a dog $\underline{\text{chases cars}}$") |
| $VP_{pl}$ | $\rightarrow$ | $V_{pl}$ | ("all dogs $\underline{\text{bark}}$") |
| $VP_{pl}$ | $\rightarrow$ | $V_{pl}$ $NP_{sg}$ | ("all dogs $\underline{\text{chase a car}}$") |
| $VP_{pl}$ | $\rightarrow$ | $V_{pl}$ $NP_{pl}$ | ("all dogs $\underline{\text{chase cars}}$") |

| | | |
|---|---|---|
| $D_{sg}$ | $\rightarrow$ | {*the, a*} |
| $D_{pl}$ | $\rightarrow$ | {*the, all*} |
| $N_{sg}$ | $\rightarrow$ | {*dog*} |
| $N_{pl}$ | $\rightarrow$ | {*dogs*} |
| $V_{sg}$ | $\rightarrow$ | {*barks, walks, sees*} |
| $V_{pl}$ | $\rightarrow$ | {*bark, walk, see*} |

- But this quickly explodes the number of rules
- And the explosion is worse with more agreement rules:

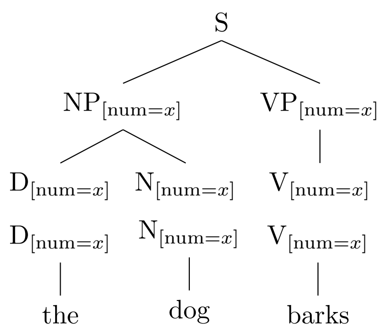
$$\begin{array}{lll}
VP_{sg,1st,masc} & \rightarrow & V_{sg,1st,masc} \\
VP_{sg,1st,fem} & \rightarrow & V_{sg,1st,fem} \\
VP_{sg,2nd,masc} & \rightarrow & V_{sg,2nd,masc} \\
VP_{sg,2nd,fem} & \rightarrow & V_{sg,2nd,fem} \\
... & & \\
VP_{pl,3rd,fem} & \rightarrow & V_{pl,3rd,fem}
\end{array}$$

- We can simplify this greatly using *feature structures* that use variables to ensure agreement

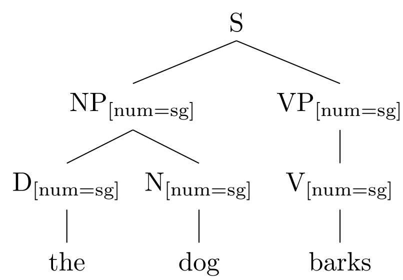
$$\begin{array}{lll}
S & \rightarrow & NP_{[\text{num}=x]}\ VP_{[\text{num}=x]} \\
 & & \\
NP_{[\text{num}=x]} & \rightarrow & D_{[\text{num}=x]}\ N_{[\text{num}=x]} \\
 & & \\
VP_{[\text{num}=x]} & \rightarrow & V_{[\text{num}=x]} \\
VP_{[\text{num}=x]} & \rightarrow & V_{[\text{num}=x]}\ NP_{[\mathbf{num}=\boldsymbol{y}]}
\end{array}$$

$$\begin{array}{lll}
D_{[\text{num}=\text{sg}]} & \rightarrow & \{\textit{a, every}\} \\
D_{[\text{num}=\text{pl}]} & \rightarrow & \{\textit{all, some}\} \\
D_{[\mathbf{num}=\boldsymbol{z}]} & \rightarrow & \{\textit{the}\} \\
 & & \\
N_{[\text{num}=\text{sg}]} & \rightarrow & \{\textit{dog}\} \\
N_{[\text{num}=\text{pl}]} & \rightarrow & \{\textit{dogs}\} \\
 & & \\
V_{[\text{num}=\text{sg}]} & \rightarrow & \{\textit{barks, walks, sees}\} \\
V_{[\text{num}=\text{pl}]} & \rightarrow & \{\textit{bark, walk, see}\}
\end{array}$$

- Feature structures must *unify* as they are combined
  - For example, all $x$s and $z$ must resolve to the same value (*sg* or *pl*)




(e) Need to unify underspecified features in the rules with secified features in the N and V terminal rules




(f) all $x$s and $z$ resolve to "sg"

- Similar features could be set up for other required agreements
  - PRO[num=sg, per=3rd, gen=masc] $\rightarrow$ *il*

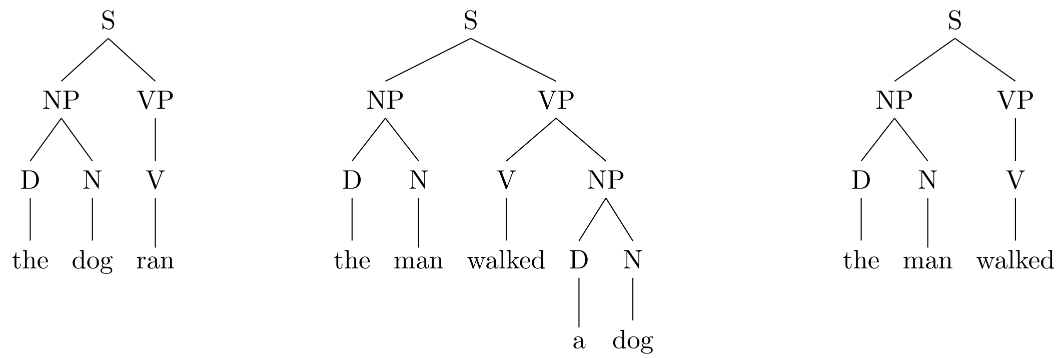
# 5 Probabilistic Context-Free Grammars (PCFG)

- A CFG is deterministic
  - It can decide whether a sentence is in the language (grammatical), or not
  - It can't judge whether one sentence is more likely than another
  - Problematic since we want to say that every sentence is *possible*, even if it's not likely
- A PCFG is a CFG in which, for every non-terminal, we have a probability distribution over possible productions
- In other words, for each non-terminal A, we have a distribution $p(\beta \mid A)$
  - The probability that, given a parent non-terminal A, we choose the production rule that yields $\beta$
  - We can alternatively write this as $p(A \rightarrow \beta)$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| S | $\rightarrow$ | NP VP | 1.0 | D | $\rightarrow$ | *the* | 0.6 |
| | | | | D | $\rightarrow$ | *a* | 0.4 |
| NP | $\rightarrow$ | D N | 0.7 | | | | |
| NP | $\rightarrow$ | N | 0.2 | N | $\rightarrow$ | *man* | 0.5 |
| NP | $\rightarrow$ | D N PP | 0.1 | N | $\rightarrow$ | *dog* | 0.4 |
| | | | | N | $\rightarrow$ | *telescope* | 0.1 |
| VP | $\rightarrow$ | V NP | 0.4 | | | | |
| VP | $\rightarrow$ | V | 0.4 | V | $\rightarrow$ | *barks* | 0.2 |
| VP | $\rightarrow$ | V NP PP | 0.2 | V | $\rightarrow$ | *walks* | 0.4 |
| | | | | V | $\rightarrow$ | *saw* | 0.4 |

Estimating parameters (MLE)

- We can calculate the MLE parameters of a PCFG model by counting productions in a corpus of parse trees
- Assume these three sentences comprise a corpus:




- We estimate by counting up all productions in the corpus and normalizing
  - Since every non-terminal $A$ must produce *something*, we can simplify slightly

$$p(\beta \mid A) = \frac{C(A \rightarrow \beta)}{\sum_{\beta' \in P} C(A \rightarrow \beta')} = \frac{C(A \rightarrow \beta)}{C(A)}$$

- Estimates from the above corpus of trees yield:

| | | | $C(A \rightarrow \beta)$ | $p(\beta \mid A)$ | | | | $C(A \rightarrow \beta)$ | $p(\beta \mid A)$ |
|---|---|---|---|---|---|---|---|---|---|
| S | $\rightarrow$ | NP VP | 3 | 1.0 | D | $\rightarrow$ | *the* | 3 | 0.75 |
| | | | | | D | $\rightarrow$ | *a* | 1 | 0.25 |
| NP | $\rightarrow$ | D N | 4 | 1.0 | | | | | |
| | | | | | N | $\rightarrow$ | *dog* | 2 | 0.5 |
| VP | $\rightarrow$ | V | 2 | 0.66 | N | $\rightarrow$ | *man* | 2 | 0.5 |
| VP | $\rightarrow$ | V NP | 1 | 0.33 | | | | | |
| | | | | | V | $\rightarrow$ | *walked* | 2 | 0.66 |
| | | | | | V | $\rightarrow$ | *ran* | 1 | 0.33 |

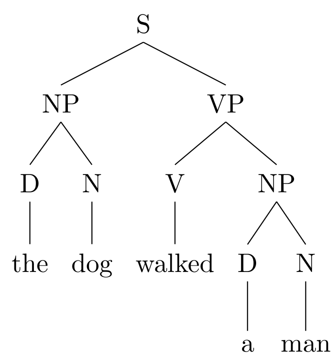
Add-$\lambda$ smoothing

- Same as always (where $P$ is the set of all possible $\beta$s produced):

$$p(\beta \mid A) = \frac{C(A \to \beta) + \lambda}{\sum_{\beta' \in P}(C(A \to \beta') + \lambda)} = \frac{C(A \to \beta) + \lambda}{(\sum_{\beta' \in P} C(A \to \beta')) + \lambda|P|} = \frac{C(A \to \beta) + \lambda}{C(A) + \lambda|P|}$$

Likelihood of a (parsed) sentence

- We can use this to calculate the likelihood of seeing a particular parse of a particular sentence
- Multiply the probabilities of all productions found in the tree
- Assume this tree:




- Count up the number of each production in the tree, and get the probability of each production from the above table

| | | | count | prob | | | | count | prob |
|---|---|---|---|---|---|---|---|---|---|
| S | $\to$ | NP VP | 1 | 1.0 | D | $\to$ | *the* | 1 | 0.75 |
| | | | | | D | $\to$ | *a* | 1 | 0.25 |
| NP | $\to$ | D N | 2 | 1.0 | | | | | |
| | | | | | N | $\to$ | *dog* | 1 | 0.5 |
| VP | $\to$ | V NP | 1 | 0.33 | N | $\to$ | *man* | 1 | 0.5 |
| | | | | | | | | | |
| | | | | | V | $\to$ | *walked* | 1 | 0.66 |

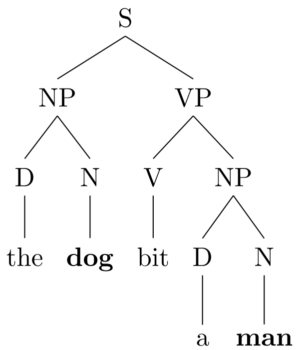
- Find the product:

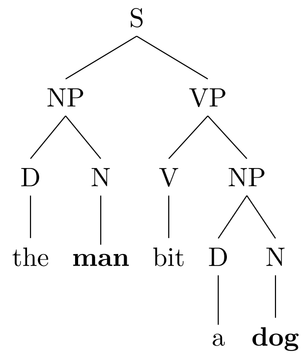
$$\begin{aligned} & p(\text{S} \to \text{NP VP}) \cdot p(\text{NP} \to \text{D N})^2 \cdot p(\text{VP} \to \text{V NP}) \cdot p(\text{D} \to \textit{the}) \cdot p(\text{D} \to \textit{a}) \cdot p(\text{N} \to \textit{dog}) \\ & \quad \cdot p(\text{N} \to \textit{man}) \cdot p(\text{V} \to \textit{walked}) \\ & = 1.0 \cdot 1.0^2 \cdot 0.33 \cdot 0.75 \cdot 0.25 \cdot 0.5 \cdot 0.5 \cdot 0.66 \end{aligned}$$

- Note that the $p(\text{NP} \to \text{D N})$ term is squared since the production "NP $\to$ D N" appears twice in the tree

# 6 Lexicalized Grammars

- Problem: trees do not take semantic coherence into account
- Production rules are independent




- These two sentence have *exactly* the same likelihood
- Since it's a product, we can swap two productions without changing the result

$$p(\text{S} \to \text{NP VP})\cdot p(\text{NP} \to \text{D N})^2 \cdot p(\text{VP} \to \text{V NP})\cdot p(\text{D} \to \textit{the})\cdot p(\text{D} \to \textit{a})\cdot \mathbf{p}(\mathbf{N} \to \boldsymbol{dog})\cdot \mathbf{p}(\mathbf{N} \to \boldsymbol{man})\cdot p(\text{V} \to \textit{bit})$$

$$\downarrow \qquad\qquad \downarrow$$

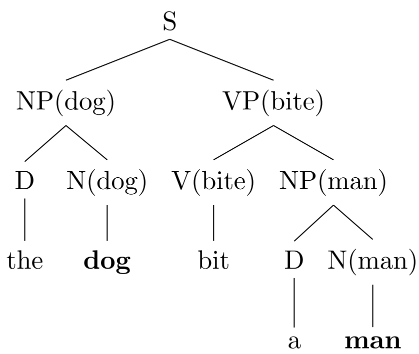
$$p(\text{S} \to \text{NP VP})\cdot p(\text{NP} \to \text{D N})^2 \cdot p(\text{VP} \to \text{V NP})\cdot p(\text{D} \to \textit{the})\cdot p(\text{D} \to \textit{a})\cdot \mathbf{p}(\mathbf{N} \to \boldsymbol{man})\cdot \mathbf{p}(\mathbf{N} \to \boldsymbol{dog})\cdot p(\text{V} \to \textit{bit})$$

- Solution: lexicalize the grammar
- Subcategorize rules with their head words:

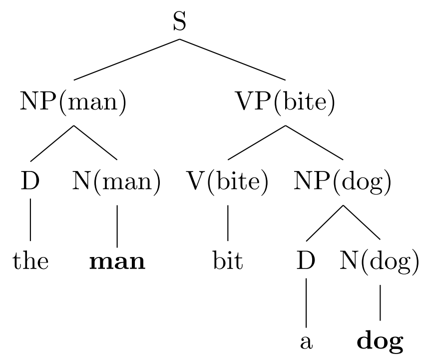
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| S | → | NP(man) VP(bite) | 0.3 | D | → | *the* | 0.6 |
| S | → | NP(dog) VP(bite) | 0.7 | D | → | *a* | 0.4 |
| S[1] | → | NP(sandwich) VP(bite) | 0.0 | | | | |
| NP(man)[2] | → | D N(man) | 0.8 | N(man)[2] | → | *man* | 0.7 |
| NP(man)[2] | → | N(man) | 0.2 | N(man)[2] | → | *men* | 0.3 |
| NP(dog)[3] | → | D N(dog) | 0.4 | N(dog)[3] | → | *dog* | 0.2 |
| NP(dog)[3] | → | N(dog) | 0.6 | N(dog)[3] | → | *dogs* | 0.8 |
| VP(bite)[4] | → | V(bite) NP(man) | 0.2 | V(bite) | → | *bite* | 0.3 |
| VP(bite)[4] | → | V(bite) NP(dog) | 0.1 | V(bite) | → | *bites* | 0.2 |
| VP(bite)[4] | → | V(bite) NP(sandwich) | 0.7 | V(bite) | → | *biting* | 0.1 |
| VP(bite)[5] | → | V(bite) | 0.0 | V(bite) | → | *bit* | 0.4 |

  - [1]Sandwiches don't bite
  - [2]Men are likely talked about in the singular
  - [3]Dogs are likely talked about in the plural
  - [4]Men are bitten infrequently, dogs are bitten less frequently, and sandwiches are bitten often.
  - [5]"Biting" can't be intransitive

- Downside: increased sparsity!




(g) more likely




(h) unlikely

# 7 Generative Model

- Like naïve Bayes models, N-Gram models, and hidden Markov models
- Two probability distribution: $p(\beta \mid \alpha)$, for production rules $\alpha \to \beta$, and $p(\sigma)$, where $\sigma$ is a possible "start" symbol
- Generative story:
  1. Choose a start symbol $x$ from the distribution over start symbols $p(\sigma)$
  2. If $x$ is a terminal, STOP
  3. Else, choose some $\beta$ from $p(\beta \mid x)$
  4. For each symbol $y$ in $\beta$, go to step 2
- For each node with symbol $x$, we choose a production rule of the form $x \to \beta$ according to their probabilities and then recursively choose rules for every node in $\beta$ until we reach terminals for all branches.

# 8 Parsing

Top-Down

- Never try to build tree that doesn't make a sentence.
- Waste time exploring trees that don't end up with the correct words.

Bottom-Up

- Never make a tree that doesn't start with the right words.
- Waste time exploring trees that don't make a sentence.

# 9 Parsing with P-CKY

Find the mostly likely parse tree $t$ for the sentence $w_0 ... w_n$

- $\hat{t} = \text{argmax}_t \; p(t \mid w_0 ... w_n)$
- Since we know how to find the likelihood of a parse (from above), we could enumerate all possible trees, and find the likelihood of each one.
- But the number of possible trees is enormous
- We will use a *dynamic programming* algorithm to find the best parse tree: P-CKY
- Probabilistic version of the CKY algorithm, which can be used with any CFG
- Analgous to the Viterbi algorithm (both are dynamic programming algorithms)

Chomsky Normal Form (CNF)

- CKY requires that the grammar be in CNF
- All productions must be one of:
  - Non-terminal producing exactly 2 non-terminals: A $\rightarrow$ B C
    Non-terminal producing exactly 1 non-terminal: A $\rightarrow$ B
    Non-terminal producing exactly 1 terminal: A $\rightarrow$ $w$
- Any CFG can be convered to CNF
  - Producing more than two nonterminals: "A $\rightarrow$ B ... Y Z" becomes
    * A $\rightarrow$ B ... [Y+Z]
      [Y+Z] $\rightarrow$ Y Z
    * Repeat as necessary until you just have A $\rightarrow$ B X
    * NP $\rightarrow$ D Adj N becomes
      NP $\rightarrow$ D [Adj+N]
      [Adj+N] $\rightarrow$ Adj N
  - For our trees we should never have terminals and nonterminals mixed in productions since all terminals a produced by unary rules from POS tags, which never produce anything other than terminals. (But if we needed to we could create dummy non-terminals for those terminals.)

P-CKY Idea

1. Make a pass over the whole sentence, trying to find subtrees that explain spans within the sentence
2. Dynamic programming: store intermediate results in a chart
3. Start with the leftmost word. For each non-terminal (NT), find the most plausible subtree that covers just that word.

4. Move to the next word

   a. Again, for each NT, find the most plausible subtree that covers just that word.
   b. Then, for each NT, find the most plausible subtree that covers both the word and the previous word.
   c. Same for the three-word span of the word and the previous two words.
   d. Continue until you have the entire span from the beginning of the sentence to the word.
   e. Then move to the next word and repeat.

5. You are finished when you have, for each NT, the most plausible tree covering the entire span of the sentence.

6. Pick the root node that is most plausible considering the *a priori* probability of an NT being the root of a sentence.

P-CKY Algorithm

- NT is the set of all non-terminals (including composites and POS tags)
- $N$ = length of sentence $w_0 ... w_{N-1}$
- `table[i,j][A]` is the best possible score for a subtree spanning words $w_i ... w_{j-1}$ with $A$ as its root
- `back[i,j][A] = (k,B,C)` indicates that the best possible subtree spanning words $w_i ... w_{j-1}$ with $A$ as its root is comprised of a left subtree spanning words $w_i ... w_{k-1}$ with $B$ as its root and a right subtree spannign words $w_k ... w_{j-1}$ with $C$ as its root.

# 10 P-CKY Example

"the complex houses married students"

NP[the complex houses] VP[V[married] NP[N[students]]]
NP[this complex] VP[V[houses] [NP[A[married] [N students]]]

# 11 N-Best Parses

Useful for providing multiple possiblities to down-stream applications.

Can be fed into a discriminative re-ranker that uses something like MaxEnt with features built from linguistic knowlege to re-score the parses. The discriminative model wouldn't be used for *parsing*, just to calculate the likelihoods of the parses that were found.

# 12 Other Grammatical Formalisms

## 12.1 TAG: Tree-Adjoining Grammar

- http://www.seas.upenn.edu/~joshi/joshi-schabes-tag-97.pdf
- Instead of single-layer production rules of a CFG, production rules are tree *fragments*
- Fragments have "incomplete" nodes that must produce to other tree fragments. (Just like how "incomplete" nodes in a CFG must produce the next rule.)
- Lexicalized: Each fragment centered around a word
- Mildly context-sensitive (as opposed to context-free)

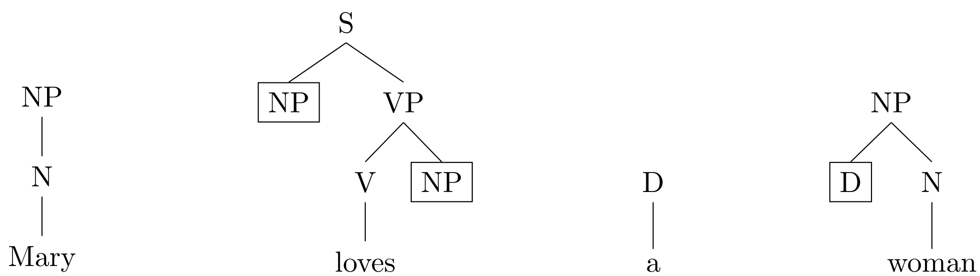



Figure 1: TAG productions

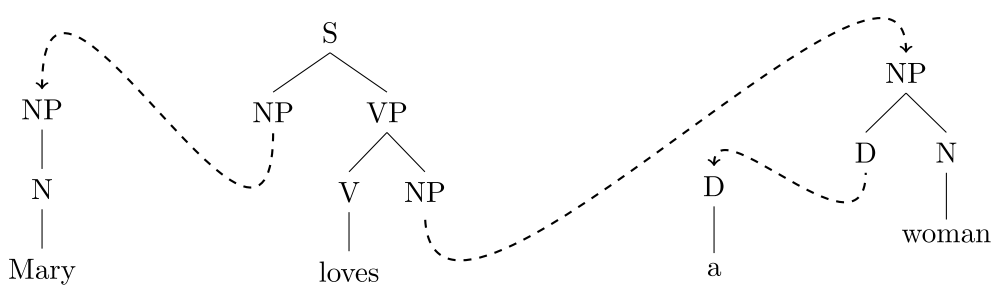



Figure 2: TAG fragment combination giving parse for "Mary loves a woman"

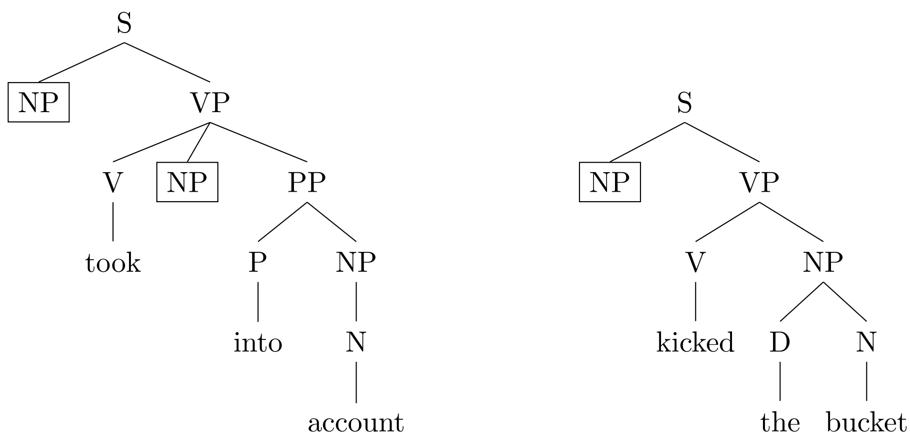



Figure 3: TAG productions for common phrases

### 12.2 CCG: Combinatory Categorial Grammar

- Instead of atomic POS tags and Non-Terminals, use *categories* that are constructed from other categories:
  - Categories are defined by a (context-free) grammar
    atomic categories: C $\rightarrow$ { S, NP, N }
    complex categories: C $\rightarrow$ { C/C, C\C }
  - Slash and backslash operators indicate that the category is a function:
    * A/B is a category that looks directly to its **right** for something of category B, and combines with it to produce an A
    * A\B is a category that looks directly to its **left** for something of category B, and combines with it to produce an A
    * So B is the *input* to the function, and *A* is the output
- Mildly context-sensitive
- Weakly equivalent to TAG

Words are assigned categories in the *Lexicon*:

Mary: NP

sleeps: S\NP

loves: (S\NP)/NP

a: NP/N

woman: N

The parse tree for a sentence, then, is the result of doing the combinations:

### 12.3 Dependency Parsing


Figure 4: Series of CCG combinations to parse “Mary sleeps”

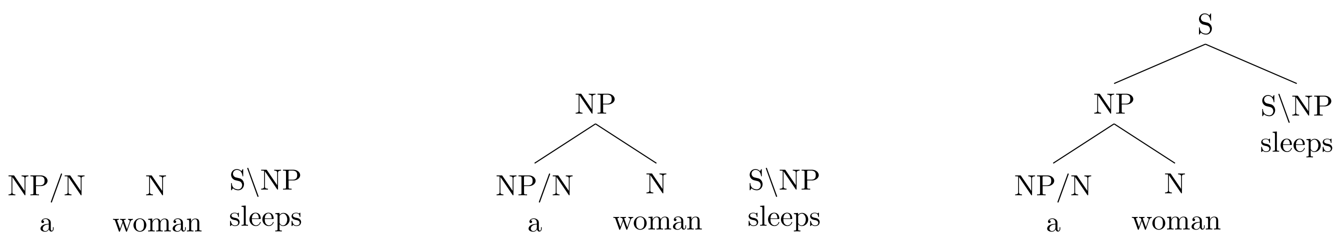



Figure 5: Series of CCG combinations to parse “a woman sleeps”

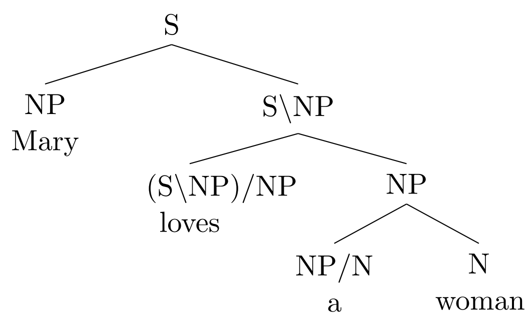
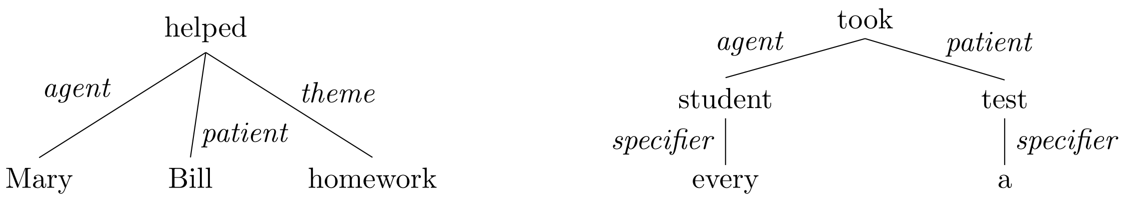



Figure 6: CCG parse tree for “Mary loves a woman”




(a) “Mary helped Bill with homework.” (b) “Every student took a test.”

Figure 7: Dependency parse trees

**Algorithm 1:** P-CKY

```
Initialize table to be an N+1 x N+1 array (for (best-so-far) scores)
Initialize back to be an N+1 x N+1 array (for backpointers)

for j ← 1 to N do
    // Fill in the bottom of the chart (single-word spans):
    i = j − 1 ;
    for A ← NT (or restrict to just POS tags if convenient) do
        // A is a potential POS tag for w_i
        if p(w_i | A) > 0.0 then
            // A → w_i is a valid rule
            table[i,j][a] = p(w_i | A)
            back[i,j][a] = w_i

    // Fill in the higher levels of the chart (multi-word spans)
    for i ← j-1 downto 0 do
        // Binary Rules (with k as pivot)
        for k ← i+1 to j-1 do
            for B ← table[i,k] and C ← table[k,j] do
                // B is the root of a sub-tree covering the span w_i...w_{k−1}
                // C is the root of a sub-tree covering the span w_k...w_{j−1}
                for A ← NT (or restrict to just non-POS tags if convenient) do
                    // A is a potential root for the subtree spanning w_i...w_{j−1}
                    if p(B C | A) > 0.0 then
                        // A → B C is a valid rule
                        s = p(B C | A) · table[i,k][B] · table[k,j][C]
                        if table[i,j] doesn't contain A OR table[i,j][A] < s then
                            // Either we haven't seen a valid subtree coverting i to j with root A,
                            //   or this new subtree has a higher score
                            table[i,j][A] = s
                            back[i,j][A] = (k, B, C)

        // Unary Rules
        done = false
        while !done do
            done = true
            for B ← table[i,j] do
                // B is the root of a sub-tree covering the span w_i...w_{j−1}
                for A ← NT (or restrict to just non-POS, non-compound tags if convenient) do
                    // A is a potential root for the subtree spanning w_i...w_{j−1}
                    if p(B | A) > 0.0 then
                        // A → B is a valid rule
                        s = p(B | A) · table[i,j][B]
                        if table[i,j] doesn't contain A OR table[i,j][A] < s then
                            // Either we haven't seen a valid subtree coverting i to j with root A,
                            //   or this new subtree has a higher score
                            table[i,j][A] = s
                            back[i,j][A] = B
                            done = false  // if we added a new NT, then re-check
```

**Algorithm 2:** Retrieve the best-parse tree from the backpointer table

*// Select the best root element while considering the prior over all potential root elements*

root = $\underset{S \leftarrow \texttt{table[0,N]}}{\text{argmax}}$ `table[0,N][S]` $\cdot\, p(S)$

**if** *`table[0,N][root]`* $\cdot\, p(root) = 0.0$ **then**

  *// There is no valid parse of the sentence*

**else**

  FollowBackpointers(root, 0, N)

**Function** FollowBackpointers(A, i, j) : Tree

  `back[i,j][A]` **match**

    **case** (k, B, C) $\Rightarrow$ // Binary branch

      Tree(A, FollowBackpointers(B, i, k), FollowBackpointers(C, k, j))

    **case** B $\Rightarrow$ // Unary branch

      Tree(A, FollowBackpointers(B, i, j))

    **case** $w$ $\Rightarrow$ // Terminal (word)

      Leaf($w$)