FNLP - Week 6: Syntactic Parsing

Antonio León Villares

March 2022

Contents

Week 5 involved strikes, so no lectures were released

1 Syntax

1.1 The Need for Syntax

- **What is syntax?**
  - the way in which **words** are **arranged together**
- **Why is syntax important?**
  - necessary to develop an **accurate** model of **language**
  - BOW, N-Gram and HMMs not good enough, since they rely on a **fixed-length** history
  - for example,

    "Looking at the amazing view, he couldn't help but gasp"

    a trigram model would have to predict "gasp", given "help but" - unlikely for this to be successful; however, we could have easily predicted it
  - **long-range dependencies** are important for a **language model**: words depend on each other, independently of many intervening words between them:

    Sam/Dogs sleeps/sleep soundly
    Sam, who is my cousin, sleeps soundly
    Dogs often stay at my house and sleep soundly
    Sam, the man with red hair who is my cousin, sleeps soundly

- **What is a theory of syntax?**
  - theory explaining which sentences are **grammatical/well formed**
  - this need not mean that a sentence is **meaningful** (i.e "Colourless green ideas sleep furiously" is grammatical, but doesn't make sense)
  - the 2 (main) theories of syntax are:
    - **constituency structures**
    - **dependency structures**

1.2 Constituents

- **What is a constituent?**
  - a **group** of words (potentially a single one), which may behave as a **single unit**
  - for example, **noun phrases**
- **How can we test if a group of words is a constituent?**
  1. **Substitutability** We can "swap" constituents of the same type to produce well-formed phrases:

     Dogs sleep soundly
     My next-door neighbours sleep soundly
     Green ideas sleep soundly

     Figure 1: Notice, for example "My" can't be swapped, since "My sleep soundly" doesn't make sense.

     This more generally applies to POS categories (i.e we can swap 2 adjectives, and a phrase will still make sense)

2. **Preposed/Postposed Constructions** A constituent can be placed at different places of a phrase, with the phrase still making sense:

   ***On September seventeenth**, I'd like to fly from Atlanta to Denver.*

   *I'd like to fly from Atlanta to Denver **on September seventeenth**.*

   *I'd like to fly **on September seventeenth** from Atlanta to Denver .*

   However, the same thing won't apply if for example we use the individual words:

   ***On** I'd like to fly **September seventeenth** from Atlanta to Denver .*

3. **Coordination** We can **coordinate** constituents of the same type with conjunctions (and, or, but)

   - **Pass the test:**

     Her friends from Peru went to the show.
     Mary *and* her friends from Peru went to the show.

     Should I go through the tunnel?
     Should I go through the tunnel *and* over the bridge?

   - **Fail the test**

     We peeled the potatoes.
     *We peeled the and washed the potatoes.

4. **Clefting** Only consitutents can appear in:

   ******* *is/are who/what/where/when/why/how . . .*

   - **Pass the test:**

     They put the boxes in the basement.
     In the basement *is where* they put the boxes.

   - **Fail the test**

     They put the boxes in the basement.
     *Put the boxes is what they did in the basement.

- **What is a constituent tree?**
  - a **tree** which breaks down a sentence into its **constituents**

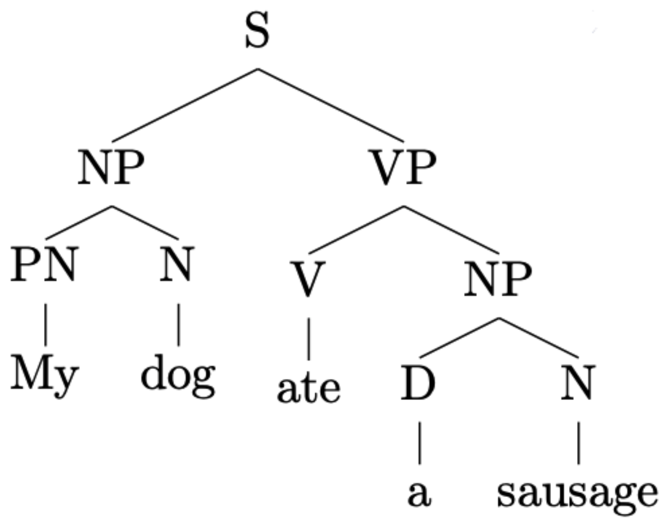


Figure 2: The internal nodes are **phrases** (i.e noun phrases like "a sandwich"), whilst the nodes immediately above words correspond to POS tags

1.3 Context Free Grammars

- **What is the structure of context free grammars?**
  - an example of a **constituency structure**: it is built from constituents
  - CFGs are (formally) a **4-tuple**:
    1. **N**: set of **non-terminal symbols** (i.e *NP* to represent a noun phrase)
    2. **Σ**: set of **terminal symbols**, disjoint from N (i.e words like "flight")
    3. **R**: set of **productions** of the form:

$$A \to \beta, \qquad A \in N, \beta \in \Sigma$$

    4. **S**: a **start symbol**

$V = \{S, VP, NP, PP, N, V, PN, P\}$

$\Sigma = \{girl, telescope, sandwich, I, saw, ate, with, in, a, the\}$

$S = \{S\}$

$R:$

$S \rightarrow NP \;\; VP$	(NP A girl) (VP ate a sandwich)	$N \rightarrow girl$
		$N \rightarrow telescope$
$VP \rightarrow V$		$N \rightarrow sandwich$
$VP \rightarrow V \;\; NP$	(V ate) (NP a sandwich)	$PN \rightarrow I$
$VP \rightarrow VP \;\; PP$	(VP saw a girl) (PP with a telescope)	$V \rightarrow saw$
		$V \rightarrow ate$
$NP \rightarrow NP \;\; PP$	(NP a girl) (PP with a sandwich)	$P \rightarrow with$
$NP \rightarrow D \;\; N$	(D a) (N sandwich)	$P \rightarrow in$
$NP \rightarrow PN$		$D \rightarrow a$
$PP \rightarrow P \;\; NP$	(P with) (NP with a sandwich)	$D \rightarrow the$

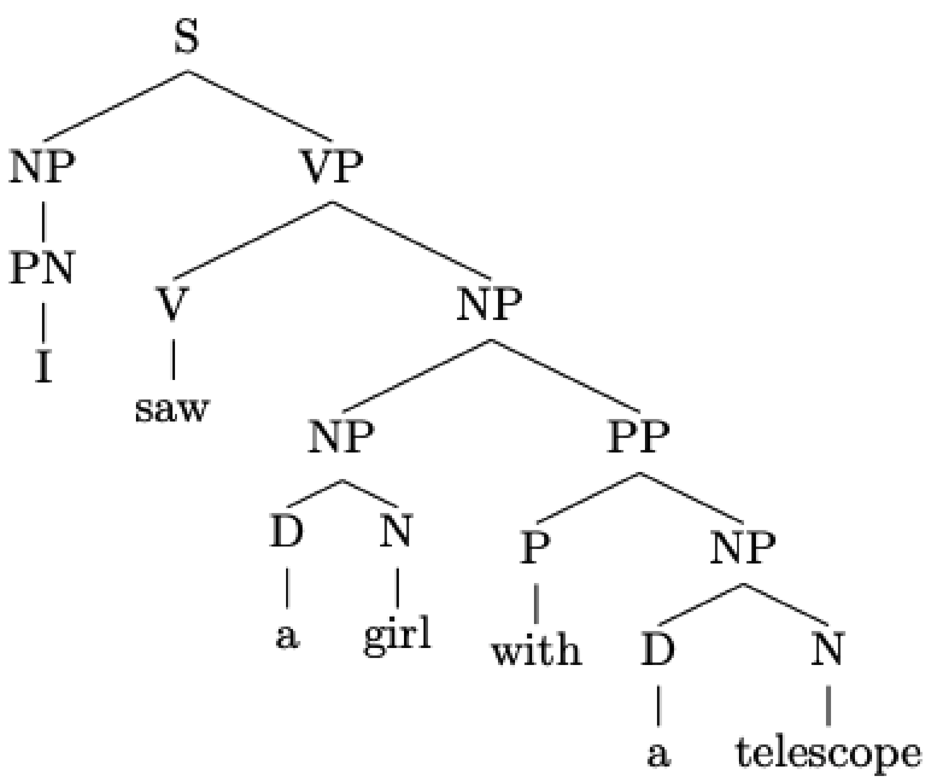


Figure 3: A possible **constituent tree** derived from the CFG above.

- **What is a derivation?**
  - a set of strings which can be produced from a CFG
  - can be represented using a parse tree
- **What are treebanks?**
  - **corpora** in which **sentences** are annotated using a **parse tree**
- **What types of equivalences can arise from different grammars?**
  - **Strong equivalence**: generate same set of strings **and** assign same phrase **structure** to each sentence
  - **Weak equivalence**: generate same set of strings (but different phrase structure assignment)
- **Why are these grammars called "context free"?**
  - the **production rules** are applicable independent of context

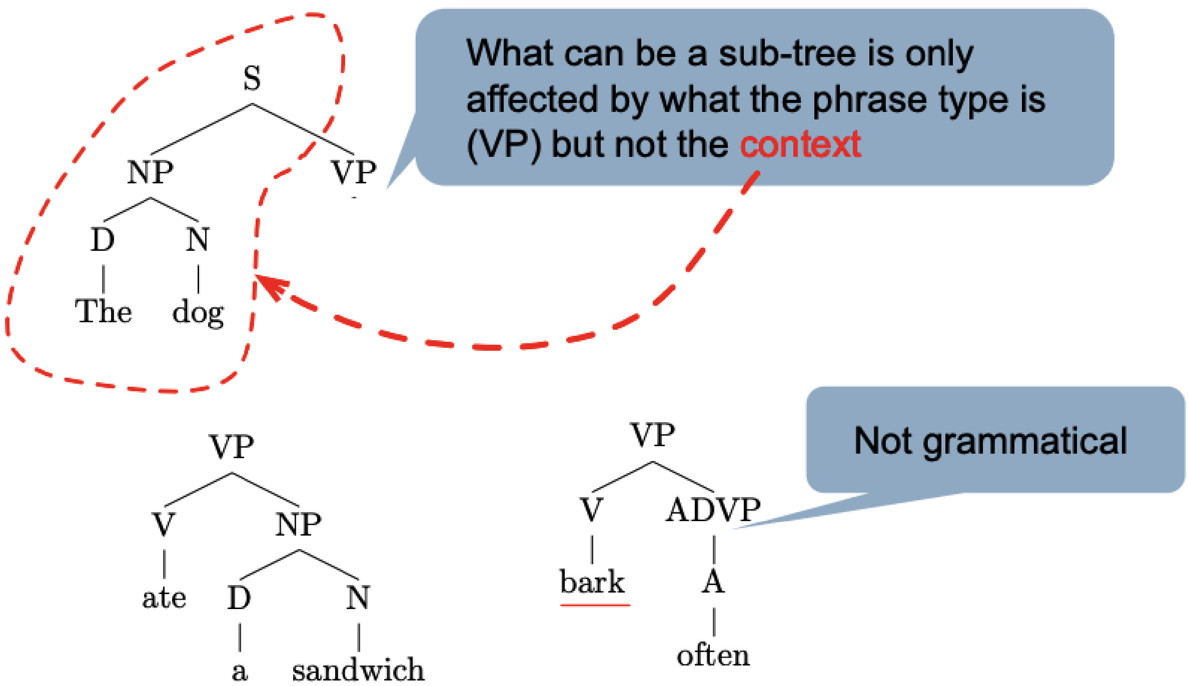


Figure 4: Here, "The dog" can be generated, without worrying about what comes after. For example, "The dog ate a sandwich" is perfectly valid.

1.4 Chomsky Normal Form

- **What format do grammars in Chomsky Normal Form take?**
  1. no ε productions (i.e no production rule can have the form $A \to \varepsilon$)
  2. a **production** can only have the following forms:

$$A \to a$$

$$A \to B\ C$$

where a is a **terminal**, and A, B, C are **non-terminals**

  - in particular, grammars in **CNF** ensure **binary branching** (except at the terminal nodes)
- **Why are CNFs important?**

- any grammar can be **converted** to a **weakly equivalent** CNF (so same language generated, but different syntactic tree)
- CNFs are the grammars on which the **CYK Algorithm** functions

- **How can a grammar be converted to CNF?**
  - rules which produce more than 2 non-terminals can be changed:

$$A \to B\ C\ D$$

    gets converted to:

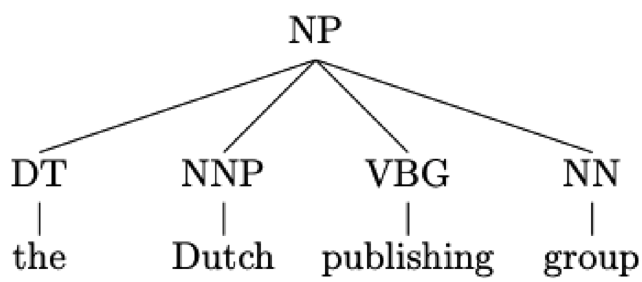
$$A \to B\ X \qquad X \to C\ D$$

  - **unary rules** can be converted to produce terminals:

$$C \to C_1 \implies C \to c_1$$

  - you also remove ε productions (but not relevant to this course)

- Consider $NP \to DT\ NNP\ VBG\ NN$



- How do we get a set of binary rules which are equivalent?

$$NP \to DT\ X$$
$$X \to NNP\ Y$$
$$Y \to VBG\ NN$$

- A more systematic way to refer to new non-terminals

$$NP \to DT\ @NP|DT$$
$$@NP|DT \to NNP\ @NP|DT\_NNP$$
$$@NP|DT\_NNP \to VBG\ NN$$

Figure 5: The format of the latter representation is useful, since it allows an easy reverse conversion for post processing.

2 Syntactic Parsing

2.1 The Purpose of Parsers

- **What is syntactic parsing?**

- the process of mapping a **sequence of words** to its **parse tree**
- getting this structure allows us to **interpret meaning**

- **Why is parsing important?**
  - correct structure $\implies$ correct meaning
  - **efficiency**: impossible to search all possible structures which match a sequence of words

- **What are the 2 fundamental properties of parsers?**
  1. **Directionality**: how is the parse tree built?
     - top-down: from S to terminals
     - bottom-up: from terminals to S
     - mixed: i.e start from left corner
  2. **Search Strategy**: how do we explore the space of possible parse trees, as to find the parse tree fitting our word sequence?

2.2 The Issue with Structural Ambiguity

- **Why is parsing hard?**
  - typical sentences can have immense **parse trees**
  - most importantly is the issue of **structural ambiguity**: how a given sequence can have **several possible parse trees**

- **How does structural ambiguity present itself?**
  1. **Attachment Ambiguity** Arises from the fact that **constituents** can be "attached" to the parse tree at different places. For example:

     "One morning I shot an elephant in my pajams"

     represents **PP-attachment ambiguity**: we don't know if the **prepositional phrase** "in my pajamas" attaches to:
     - "I": the person who shot was wearing pajamas at the time of the shooting
     - "an elephant": the elephant which got shot was wearing the shooter's pajamas
  2. **Coordination Ambiguirty** Arises from the fact that conjunctions can be applied in different ways. For example:

     "old men and women

     could refer to a group of old men, alongside women; or a group of both old men and old women.

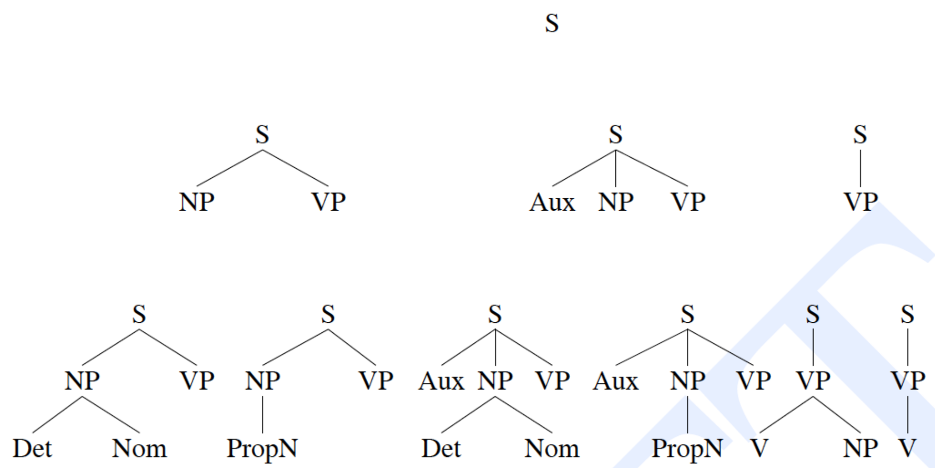
- **Is parsing unambiguous sentences easy?**
  - sentences might be unambiguous, but still hard to parse
  - this is due to **local ambiguity**: a part of a sentence is itself ambiguous, even if the whole sentence isn't
  - for example:

    "Book that flight"

    is unambiguous, but "book" is (it can be a verb or a noun), so a parser would have to consider both possible parsers, until it reaches the end of the parsing

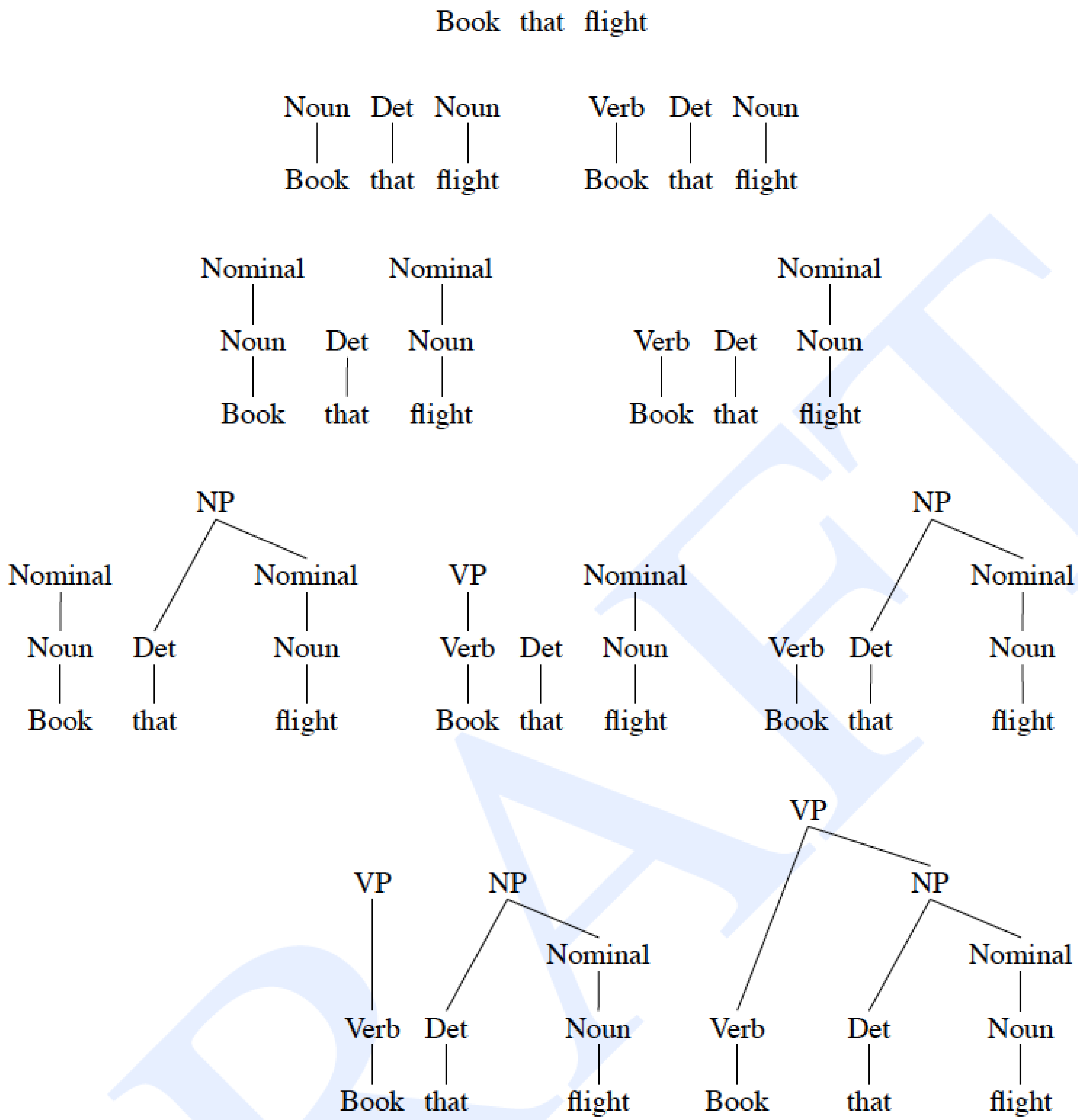
2.3 Top-Down Parsing

- **How does top-down parsing search the parse tree space?**
  - start with S nodes
  - choose one of the children to continue exploring
  - repeat with the node's child, until we reach a suitable parse tree
- **What is depth first search?**
  - go down a branch of the space as far as possible
  - if we reach an impossible parse tree, **backtrack**
- **What is breadth first search?**
  - expand all branches in parallel
  - generally not good: the number of branches is too large, so will take a long time to find a suitable parse tree
- **What is best first search?**
  - define a scoring function
  - score each partial parse, exploring the highest scoring option next



2.4 Bottom-Up Parsing

- **What is bottom-up parsing?**
  - begin with words in the sequence
  - try to build a parse tree which fits the word sequence structure
  - successful parse if S is reached



- **How do top-down and bottom-up parsing compare?**
  - top-down never explores parse trees which can't be grammatical
  - however, bottom-up explores parse trees which will always lead to parsing the sequence

2.5 The CYK Algorithm

- **What is the CYK algorithm?**
  - efficient (**dynamic programming**) bottom-up parser for CFGs
  - it applies to:
    - the **recognition problem** (is a sentence derived by a CFG?)
    - the **parsing problem** (what is the derivation tree of the sentence?)

- it relies on grammars being in CNF form
- My IADS notes on CYK and CNF conversion

- **What problems does CYK solve?**
  - **large search space**: instead of exploring the whole search space (potentially recomputing the same parse trees), it stores partial parses
  - **ambiguity**: stores **all** possible parses, so reduces the ambiguity problem

- **What is the recursive idea in CYK?**
  - the parse of a string depends on the parse of its component substrings
  - for example, to parse "Book the flight through Houston", we need to consider whether we can parse "Book" and "the flight through Houston"
  - as a **base case**, we will be parsing individual words (i.e using POS tags)

- **What table structure does CYK employ?**
  - consider a string with n words; we store results in an $n \times n$ table (rows from 0 to $n-1$, columns from 1 to n)
  - element (i, j) contains the **partial parse** (if any) of the substring $span(i, j)$

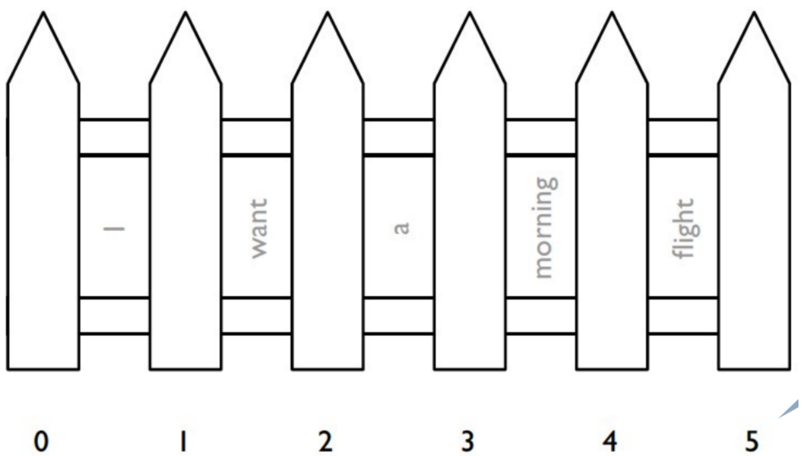


Figure 6: For example, $span(0, 1)$ = "*I*", whilst $span(3, 5)$ = *"morning flight"*.

  - ultimately, we want to know whether entry $(0, n)$ contains S as a partial parse
  - it must be noted that we only use the upper triangular part of the table (since we require that $i > j$)
  - the table is filled in **top to bottom** and **left to right**, to ensure that all possible substrings are parsed beforehand

- **What is the full CYK algorithm?**
  - from the book:

function CKY-PARSE(*words, grammar*) **returns** *table*

```
for j ← from 1 to LENGTH(words) do
    table[j−1, j] ← {A | A → words[j] ∈ grammar }
    for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
            table[i,j] ← table[i,j] ∪
                          {A | A → BC ∈ grammar,
                               B ∈ table[i,k],
                               C ∈ table[k,j] }
```

Figure 7: Technically, this only recognises whether a word sequence is well-formed; to get the parse tree, we just need to ensure that each entry is paired with a **pointer**, indicating where it was derived from.

- in lectures, the table is slightly different (it is $n \times n \times n$ and boolean, with entry (i, j, C) indicating whether $span(i, j)$ can be parsed as C or not)

```
for each wi from left to right
  for each preterminal rule C -> wi
    chart[i - 1][i][C] = true
```

Figure 8: This is how preterminal rules (i.e terminal productions) are handled.

```
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
```

Figure 9: This is how binary rules are handled. Notice, we can see the runtime will be $\mathcal{O}(n^3|R|)$, where R is the set of all productions in the grammar.

new bounds!

```
for each max from 1 to n
  for each min from max - 1 down to 0
    // First, try all binary rules as before.
    ...
    // Then, try all unary rules.
    for each syntactic category C
      for each unary rule C -> C1
        if chart[min][max][C1] then
          chart[min][max][C] = true
```

Figure 10: If we define a CNF which allows unary productions (i.e $C_1 \to C_2$), we only need a slight modification.

- **What does the CYK algorithm not account for?**
  - the algorithm can fail if there are **chains of rules**: that is, if we have $A \to B$ or $B \to C$, then $A \to B \to C \implies A \to C$ is a perfectly valid production
  - however, CYK won't account for this
  - the algorithm could be adjusted (i.e run repeatedly until entries don't change)
  - in practice, extend the grammar to enforce **transitive closure** (i.e include a rule $A \to C$ as part of the grammar)
  - **or just make sure that there are no unary rules, that is, ensure you work with a CNF, and you won't get these dumb problems**

2.5.1 Worked Example: CYK

We consider parsing:

"lead can poison"

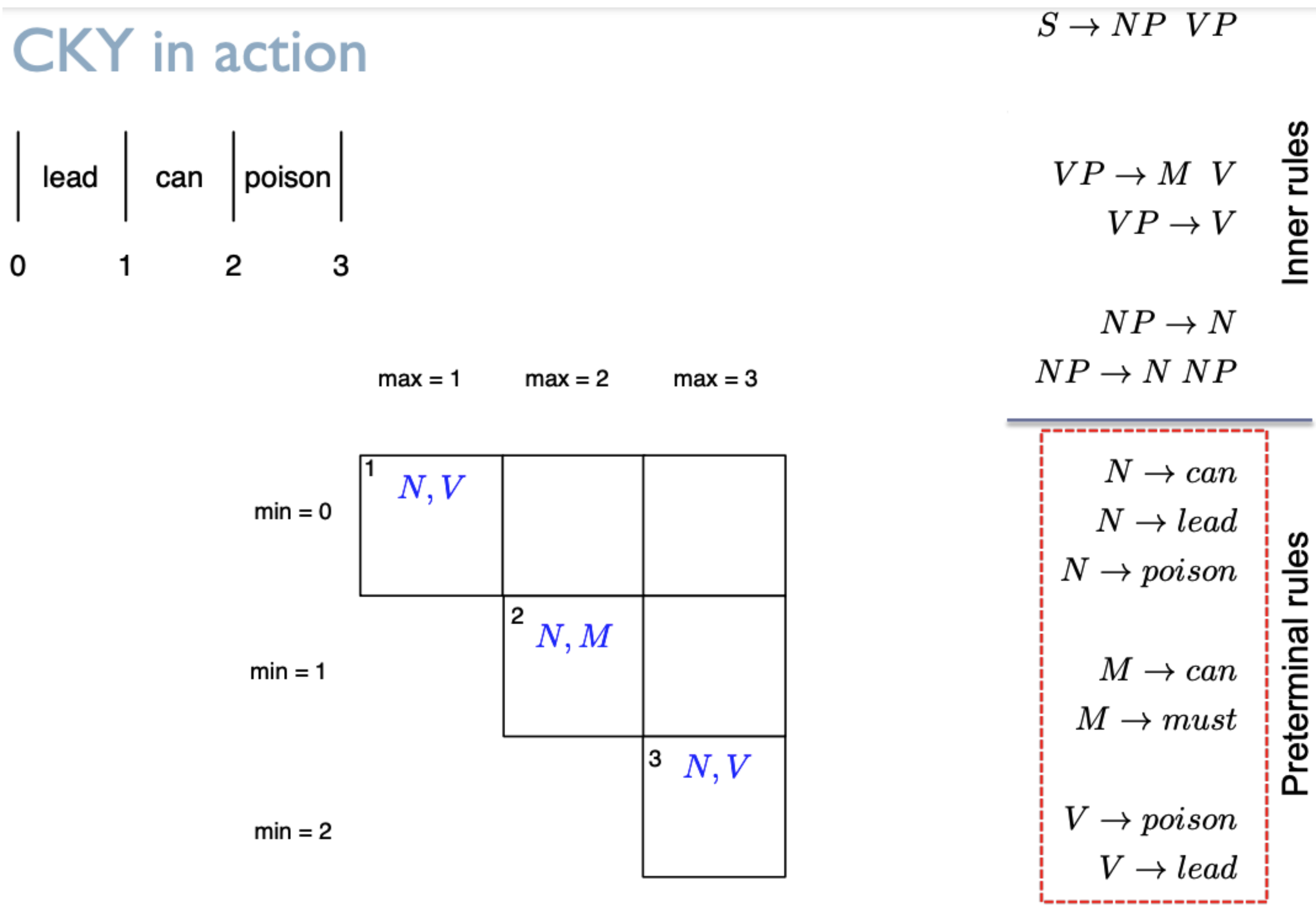


Figure 11: We begin by considering the possible derivations for words. We can first think of them as POS tags.

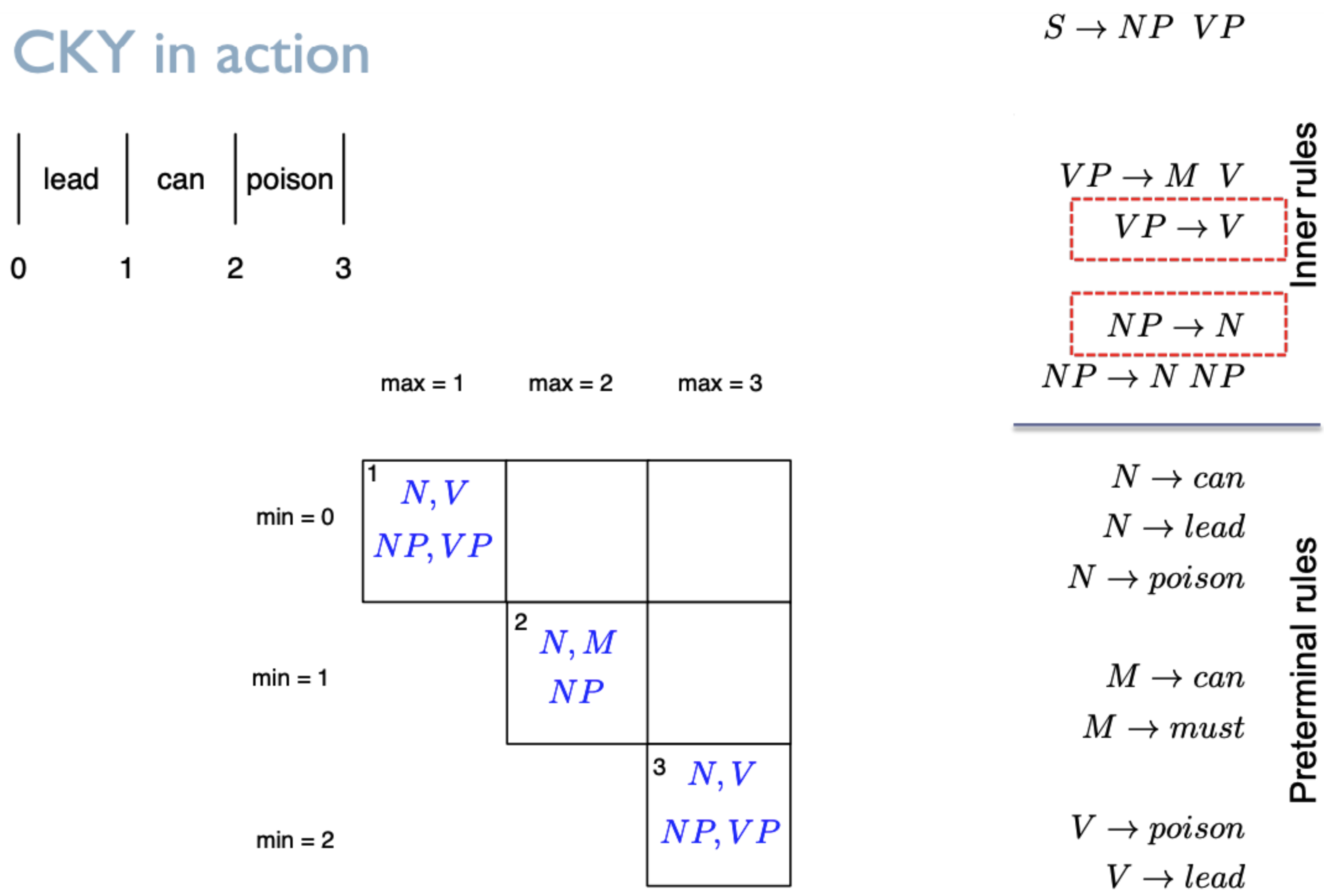


Figure 12: For some reason, they also consider unary rules, so in this case we also need to consider them. In particular, since V is a possible tag for "lead", and there is a production $VP \to V$, VP is also a valid parse for "lead".

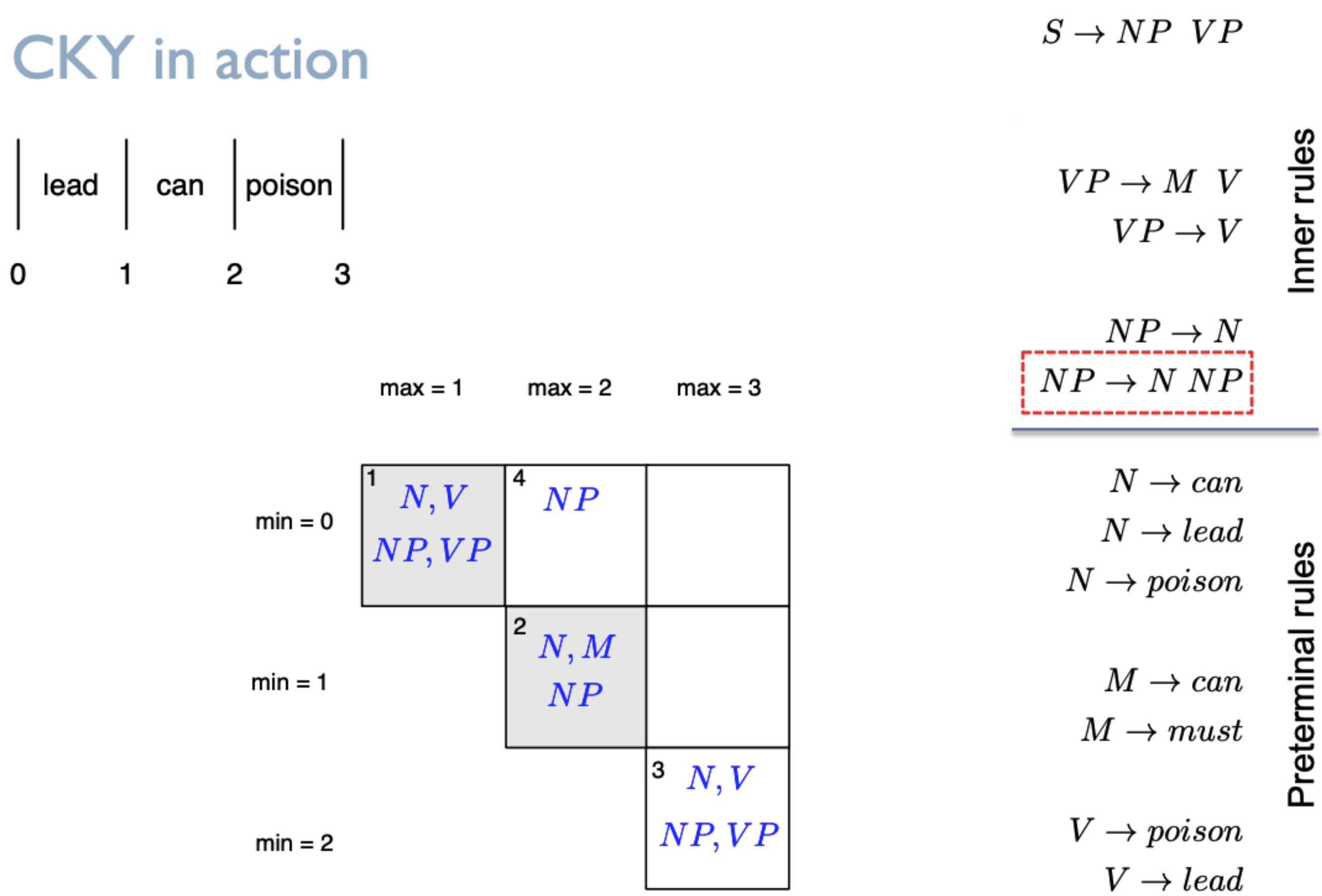


Figure 13: We now consider parsing "lead can". This relies on combining the parses of "lead" and "can". The only possible combination which has a valid production is $NP \rightarrow NNP$, so the only possible label is NP.

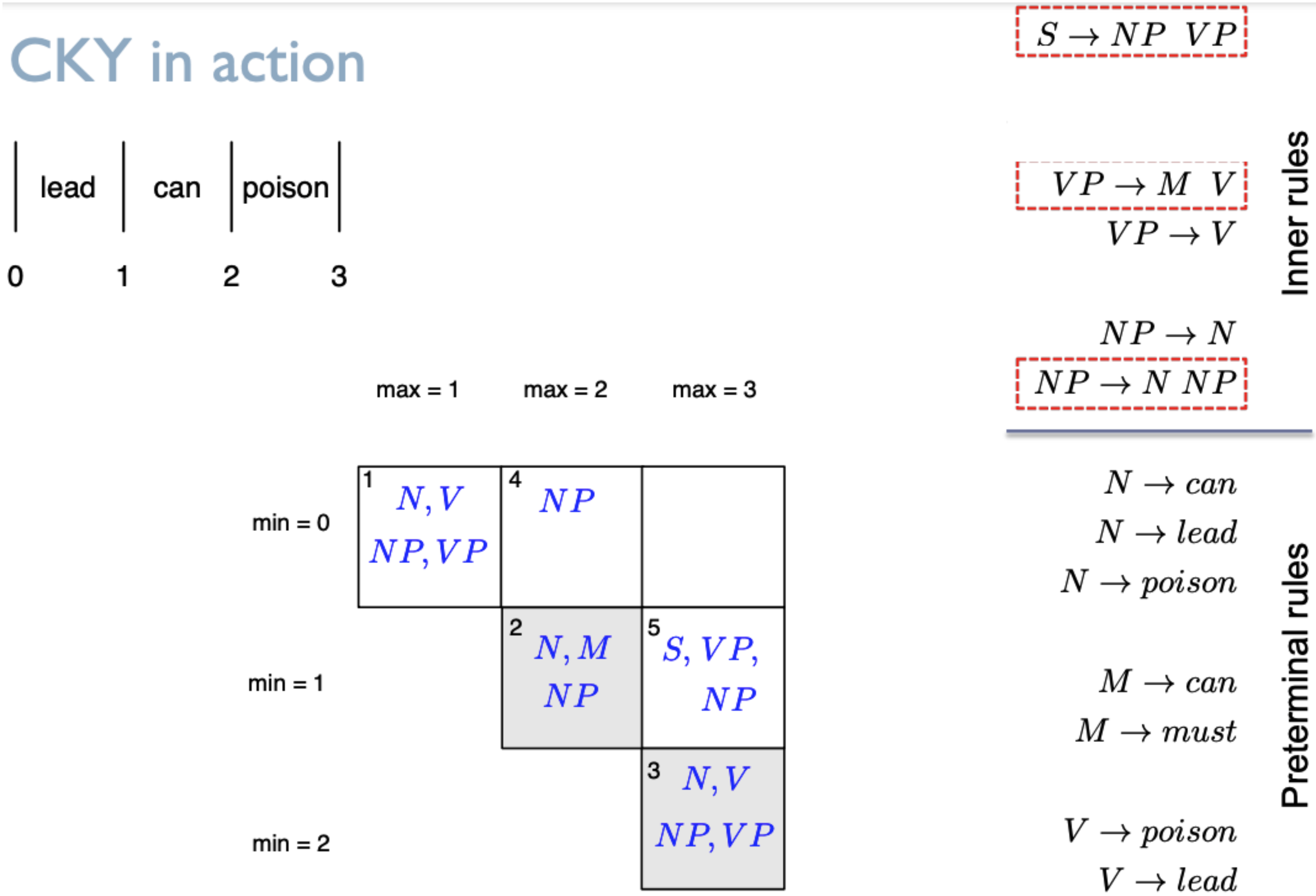


Figure 14: Similarly, for "can poison", we consider the parses of "can" and "poison". We can see there are 3 possible productions: $S \rightarrow NPVP$, $VP \rightarrow MV$ and $NP \rightarrow NNP$.

Now for "lead can poison", we need to consider 2 types of parses: "lead" + "can poison" and "lead can" + "poison".

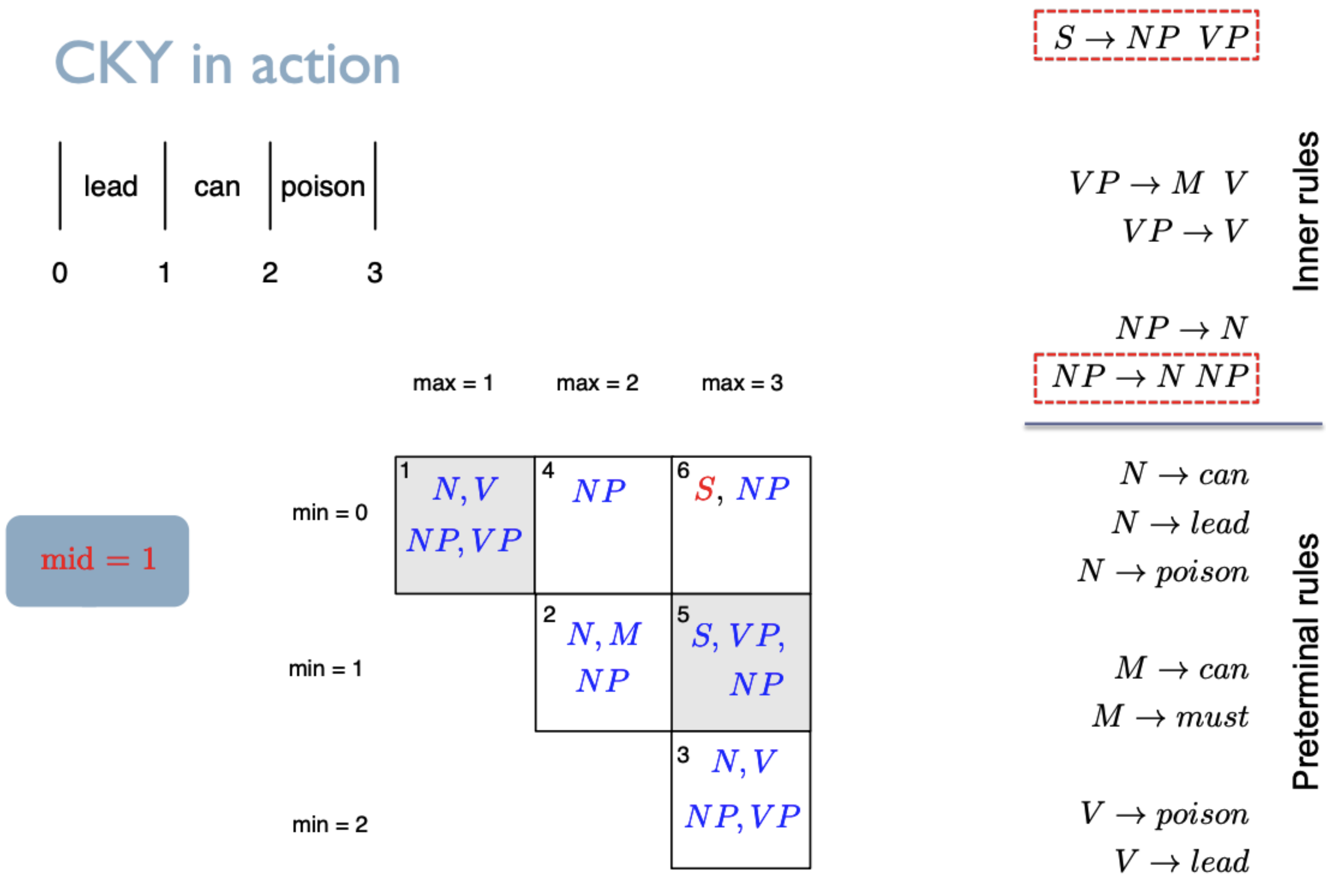


Figure 15: For "lead" + "can poison", there are only 2 possible productions: $S \rightarrow NPVP$ and $NP \rightarrow NNP$.

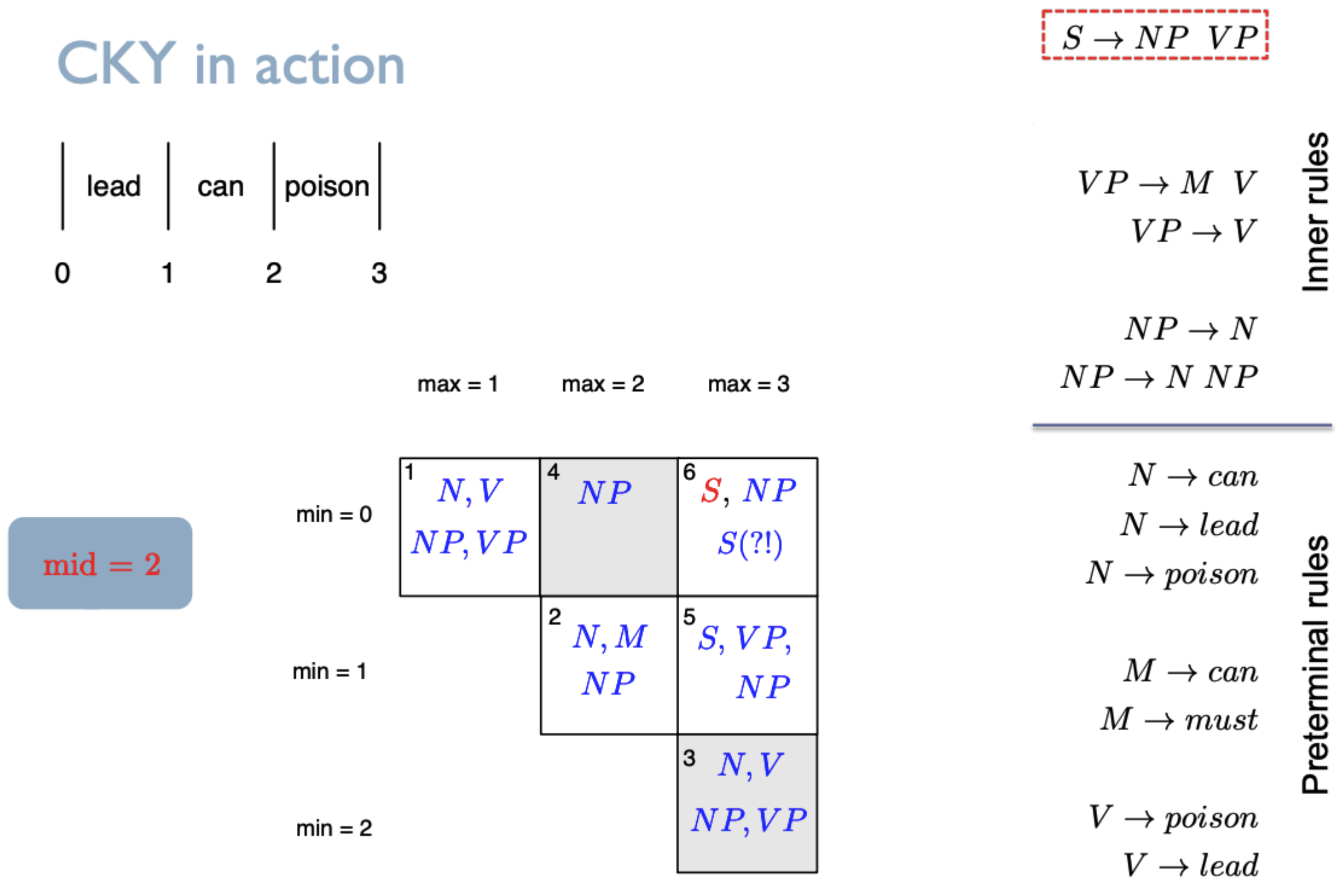


Figure 16: For "lead can" + "poison", there is only 1 possible production: $S \to NPVP$.

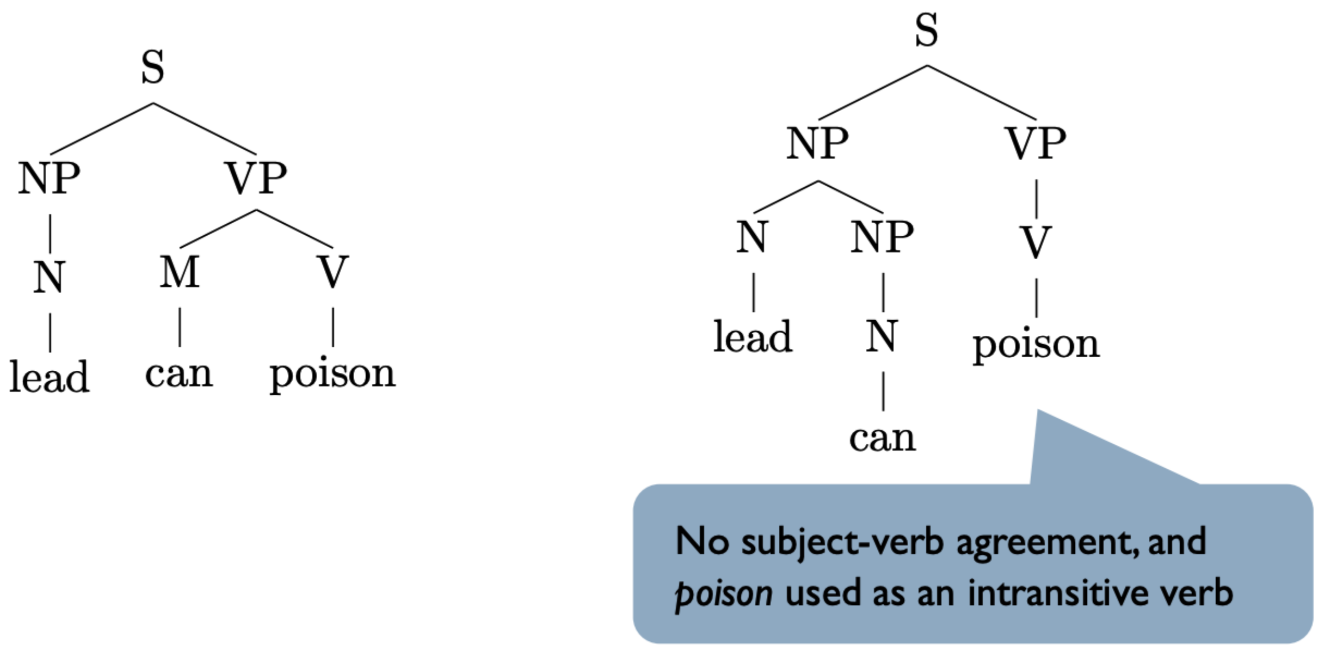


Figure 17: Hence, this sentence is ambiguous in the grammar, since it has 2 possible parse trees (but the second one is less likely - hint for the next section).

3 Statistical Parsing

3.1 Probabilistic Context Free Grammars

- **What are PCFGs?**
  - a natural **extension** of CFGs
  - formally defined as **4-tuples**:
    1. **N**: set of **non-terminals**
    2. **Σ**: set of **terminal symbols**, **disjoint** from N
    3. **R**: set of **productions**:
    $$A \rightarrow \beta[p], \qquad A \in N, \beta \in \Sigma, p \in [0,1]$$
    4. **S**: a **start symbol**
  - here p is the **probability** of the production $A \rightarrow \beta$, where:
  $$p = P(A \rightarrow \beta) = P(\beta \mid A)$$
  such that:
  $$\forall A \in N, \quad \sum_{\beta : A \rightarrow \beta \in R} P(A \rightarrow \beta) = 1$$

$S \rightarrow NP\ VP$	1.0	(NP A girl) (VP ate a sandwich)
$VP \rightarrow V$	0.2	
$VP \rightarrow V\ NP$	0.4	(VP ate) (NP a sandwich)
$VP \rightarrow VP\ PP$	0.4	(VP saw a girl) (PP with …)
$NP \rightarrow NP\ PP$	0.3	(NP a girl) (PP with ….)
$NP \rightarrow D\ N$	0.5	(D a) (N sandwich)
$NP \rightarrow PN$	0.2	
$PP \rightarrow P\ NP$	1.0	(P with) (NP with a sandwich)

$N \rightarrow girl$	0.2
$N \rightarrow telescope$	0.7
$N \rightarrow sandwich$	0.1
$PN \rightarrow I$	1.0
$V \rightarrow saw$	0.5
$V \rightarrow ate$	0.5
$P \rightarrow with$	0.6
$P \rightarrow in$	0.4
$D \rightarrow a$	0.3
$D \rightarrow the$	0.7

- **Why are PCFGs useful?**
  1. **Disambiguation**: allow us to compute the **probability of parse trees**, so we can pick the **most likely parse**
  2. **Language Modelling**: allow us to compute the **probability of a sentence**
- **How is the probability of a parse tree computed?**
  - consider a sentence S with parse tree T
  - then the probability of having S parsed with T is the product of all the productions used to expand **non-terminal** nodes. In particular, if n nodes are expanded using rules:

$$LHS_i \rightarrow RHS_i, \qquad i \in [1, n]$$

    then:

$$P(T, S) = \prod_{i=1}^{n} P(RHS_i \mid LHS_i)$$

  - however, notice that:

$$P(T, S) = P(T)P(S \mid T) = P(T)$$

    where we use the fact that $P(S \mid T) = 1$ (that is, T always generates S, since it is its parse tree)
  - thus, we can talk about the **probability of a parse tree**, independently of the sentence used to generate it (this makes sense, since the preterminal probabilities are not required to compute the probability $P(T, S)$)

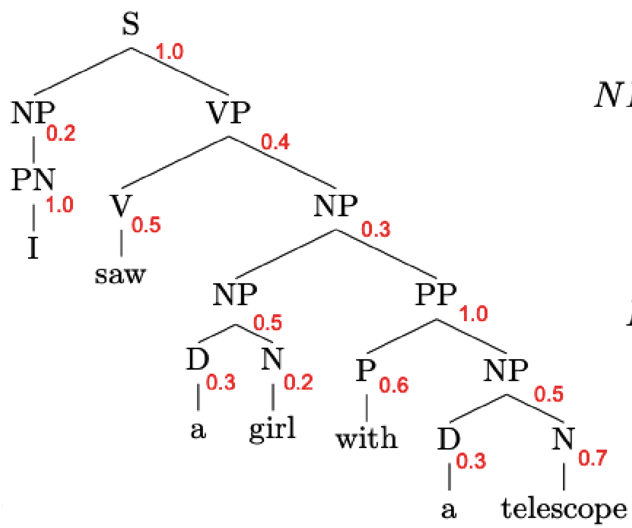
$$S \rightarrow NP\ VP \quad 1.0$$

$$VP \rightarrow V \quad 0.2$$
$$VP \rightarrow V\ NP \quad 0.4$$
$$VP \rightarrow VP\ PP \quad 0.4$$

$$NP \rightarrow NP\ PP \quad 0.3$$
$$NP \rightarrow D\ N \quad 0.5$$
$$NP \rightarrow PN \quad 0.2$$

$$PP \rightarrow P\ NP \quad 1.0$$

$$N \rightarrow girl \quad 0.2$$
$$N \rightarrow telescope \quad 0.7$$
$$N \rightarrow sandwich \quad 0.1$$
$$PN \rightarrow I \quad 1.0$$
$$V \rightarrow saw \quad 0.5$$
$$V \rightarrow ate \quad 0.5$$
$$P \rightarrow with \quad 0.6$$
$$P \rightarrow in \quad 0.4$$
$$D \rightarrow a \quad 0.3$$
$$D \rightarrow the \quad 0.7$$



$$p(T) = 1.0 \times 0.2 \times 1.0 \times 0.4 \times 0.5 \times 0.3 \times 0.5 \times 0.3 \times 0.2 \times 1.0 \times 0.6 \times 0.5 \times 0.3 \times 0.7 = 2.26 \times 10^{-5}$$

3.2 Issues with PCFGs

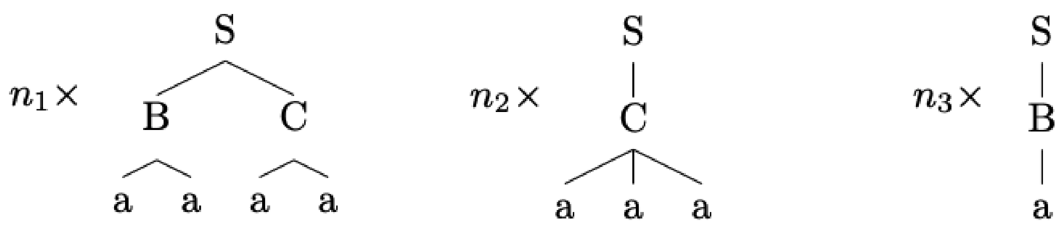
- **What are the 2 key problems of PCFGs?**
  1. they make an oftentimes invalid **independence assumption**
  2. no considerations for **lexical information**
- **Why is the independence assumption poor?**
  - PCFGs expand non-terminals **independently of context**
  - this is why we **multiply** expansion probabilities to compute the probability of a tree
  - this is an issue: for example, if an NP can be expanded in 2 ways, one expansion will be more likely than the other in a given context (i.e in a **subject**, $NP \rightarrow PRP$ occurs 91% of the time, but the production $NP \rightarrow PRP$ in general is much lower), but PCFGs don't capture these relationships
- **Why is lexical information important?**
  - PCFGs will be biased for more likely structures (i.e PP tends to attach to NP more often than to VP)
  - if lexical considerations were made, this probability could be more "refined" (i.e the preposition "into" has more affinity for the noun "sacks" - "Workers dumped sacks into a bin"; but the preposition "of" has more affinity for the verb "caught" - "Fishermen caught lots of fish")

3.3 Most Likely Parse: A Probabilistic Distribution Over Parse Trees

- **How are production probabilities computed?**
  - the ML estimate is:

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

    that is, the proportion of times in which the non-terminal X produced α
  - these probabilities are typically computed using a **treebank**
  - smoothing like **Good Turing** can be used (particularly helpful for preterminal productions)



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	$n_3/(n_1+n_2+n_3)$
$B \to a\ a$	n_1	$n_1/(n_1+n_3)$
$B \to a$	n_3	$n_3/(n_1+n_3)$
$C \to a\ a$	n_1	$n_1/(n_1+n_2)$
$C \to a\ a\ a$	n_2	$n_2/(n_1+n_2)$

- **When is a PCFG consistent?**
  - when the **sum** of probabilities of **all** sentences in the grammar is 1
  - this can happen with, for example, rules like $S \to S[1]$, which cause infinitely long strings
- **What is a proper distribution over parse trees?**
  - when the **sum** of probabilities of **all** trees in the grammar is 1:

$$\sum_T P(T) = 1$$

  - if we estimate probabilities using MLE, we are guaranteed a **proper** distribution

3.4 Probabilistic CYK

- **What is the best parse according to a PCFG?**
  - PCFGs allow us to define a **best** parse, as the most likely parse tree:

    $$\hat{T} = \underset{T \in G(x)}{argmax} P(T)$$

    where $G(x)$ is the set of **all** derivations of a sentence x
  - finding all possible T is exponential in nature, so we need to use **probabilistic CYK**
- **What is probabilistic CYK?**
  - extension of CYK, adapted to PCFGs in CNF
  - when converting to CNF, the production probabilities need to be adapted
  - for a sentence with n words, and a PCFG with V non-terminals, produces a $n+1 \times n+1 \times V$ table
  - entry (i, j, C) corresponds to the highest probability of $span(i, j)$ being a constituent of type C

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                                  and its probability
  for j ← from 1 to LENGTH(words) do
     for all { A | A → words[j] ∈ grammar }
       table[j−1, j, A] ← P(A → words[j])
     for i ← from j−2 downto 0 do
         for k ← i+1 to j−1 do
              for all { A | A → BC ∈ grammar,
                            and table[i,k,B] > 0 and table[k,j,C] > 0 }
                   if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
                      table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
                      back[i,j,A] ← {k,B,C}
     return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Figure 18: Again, use backpointer to keep track of where the most likely probability comes from to reconstruct the tree.

```
for each wi from left to right

  for each preterminal rule C -> wi

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

Figure 19: For handling preterminal 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
```

Figure 20: For handling binary rules. Again, to handle the unary productions, a slight modification is required.

- **How can we deal with unary closure?**
  - notice, if we have $A \to B$ and $B \to C$, adding $A \to C$ (by defining $P(A \to C) = P(A \to B) \times P(B \to C)$), will break the consistency of the PCFG
  - necessary, since a rule directly defined by $A \to C$ might have a very low probability, but $A \to B$ and $B \to C$ might be very likely
  - need to store the fact that $A \to C$ is composite (i.e don't evaluate the product $P(A \to B) \times P(B \to C)$), so that we can recover the parse tree
  - rules like $X \to X$ should have probability 1
- **Should PCFGs worry about infinite productions or loops?**
  - no, since such productions will have infinitesimal probabilities (large products)
  - since CYK selects largest probability, such situations won't be considered
- **How can probabilistic CYK be sped up?**
  1. **Basic Pruning**: only store labels with high probabilities (i.e within a factor of N of the most likely label); only consider rules which lead to trees with non-zero probabilities
  2. **Coarse-to-fine Pruning**: use simpler grammar to parse and precompute probabilities for each $span(i, j)$; then consider labels with non-negligible probabilities for the full parse