Syntax and Parsing

Modelling word behaviour

We've seen various ways to model word behaviour.

- Bag-of-words models: ignore word order entirely
- N-gram models: capture a fixed-length history to predict word sequences.
- HMMs: also capture fixed-length history, using latent variables.

Useful for various tasks, but a really accurate model of language needs more than a fixed-length history!

Long-range dependencies

The form of one word often depends on (agrees with) another, even when arbitrarily long material intervenes.

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

We want models that can capture these dependencies.

Phrasal categories

We may also want to capture **substitutability** at the phrasal level.

- POS categories indicate which *words* are substitutable. For example, substituting **adjectives**:

  I saw a red cat
  I saw a former cat
  I saw a billowy cat

- **Phrasal categories** indicate which *phrases* are substitutable. For example, substituting **noun phrase**:

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

Theories of syntax

A **theory of syntax** should explain which sentences are **well-formed** (grammatical) and which are not.

- Note that **well-formed** is distinct from **meaningful**.
- Famous example from Chomsky:
  Colorless green ideas sleep furiously
- However we'll see shortly that the reason we care about syntax is mainly for interpreting meaning.

Theories of syntax

We'll look at two theories of syntax to handle one or both phenomena above (long-range dependencies, phrasal substitutability):

- **Context-free grammar** (and variants)
- **Dependency grammar**

These can be viewed as different models of language behaviour. As with other models, we will look at

- What each model can capture, and what it cannot.
- Algorithms that provide syntactic analyses for sentences using these models (i.e., **syntactic parsers**).

Reminder: Context-free grammar

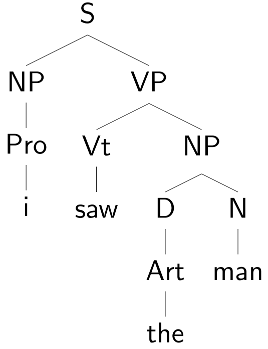
- Two types of grammar symbols:
  - **terminals** (t): words.
  - **Non-terminals** (NT): phrasal categories like $\mathtt{S}$, $\mathtt{NP}$, $\mathtt{VP}$, $\mathtt{PP}$, with $\mathtt{S}$ being the **Start symbol**. In practice, we sometimes distinguish **pre-terminals** (POS tags), a type of NT.
- Rules of the form $\mathtt{NT} \rightarrow \beta$, where β is any string of NT's and t's.
  - Strictly speaking, that's a *notation* for a rule.
  - There's also an abbreviated notation for sets of rules with same LHS: $\mathtt{NT} \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots$
- A CFG in **Chomsky Normal Form** only has rules of the form $\mathtt{NT}_i \rightarrow \mathtt{NT}_j\ \mathtt{NT}_k$ or $\mathtt{NT}_i \rightarrow \mathtt{t}_j$

CFG example

S $\rightarrow$ NP VP	**(Sentences)**
NP $\rightarrow$ D N $\mid$ Pro $\mid$ PropN	**(Noun phrases)**
D $\rightarrow$ PosPro $\mid$ Art $\mid$ NP 's	**(Determiners)**
VP $\rightarrow$ Vi $\mid$ Vt NP $\mid$ Vp NP VP	**(Verb phrases)**
Pro $\rightarrow$ i $\mid$ we $\mid$ you $\mid$ he $\mid$ she $\mid$ him $\mid$ her	**(Pronouns)**
PosPro $\rightarrow$ my $\mid$ our $\mid$ your $\mid$ his $\mid$ her	**(Possessive pronouns)**
PropN $\rightarrow$ Robin $\mid$ Jo	**(Proper nouns)**
Art $\rightarrow$ a $\mid$ an $\mid$ the	**(Articles)**
N $\rightarrow$ man $\mid$ duck $\mid$ saw $\mid$ park $\mid$ telescope	**(Nouns)**
Vi $\rightarrow$ sleep $\mid$ run $\mid$ duck	**(Intransitive verbs)**
Vt $\rightarrow$ eat $\mid$ break $\mid$ see $\mid$ saw	**(Transitive verbs)**
Vp $\rightarrow$ see $\mid$ saw $\mid$ heard	**(Verbs with NP VP args)**

Example syntactic analysis

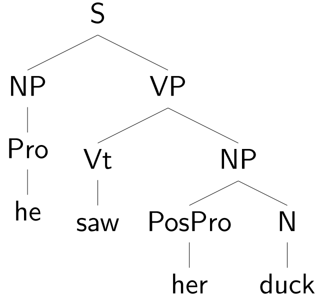
To show that a sentence is well-formed under this CFG, we must provide a parse. One way to do this is by drawing a tree:



You can think of a tree like this as *proving* that its leaves are in the language generated by the grammar.

Structural Ambiguity

Some sentences have more than one parse: **structural ambiguity**.



Here, the **structural** ambiguity is caused by **POS** ambiguity in several of the words. (Both are types of **syntactic** ambiguity.)

Attachment ambiguity

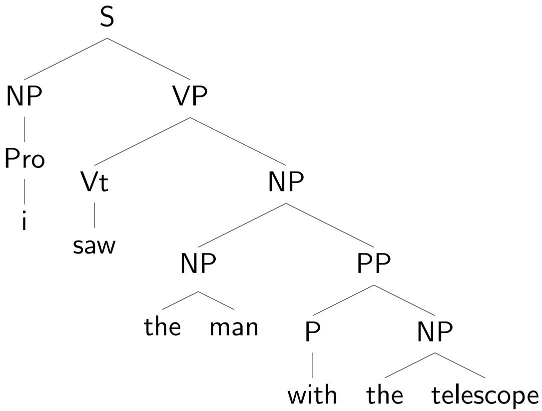
Some sentences have structural ambiguity even **without** part-of-speech ambiguity. This is called **attachment ambiguity**.

- Depends on where different phrases attach in the tree.
- Different attachments have different meanings:

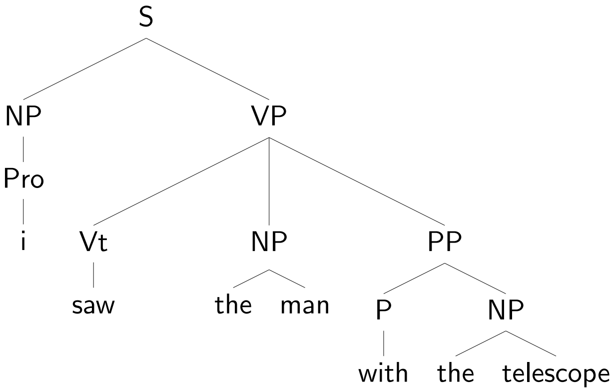
  I saw the man with the telescope
  She ate the pizza on the floor
  Good boys and girls get presents from Santa

- Next slides show trees for the first example: prepositional phrase (PP) attachment ambiguity. (Trees slightly abbreviated...)

Attachment ambiguity



Attachment ambiguity



Parsing algorithms

Goal: compute the structure(s) for an input string given a grammar.

- Ultimately, want to use the structure to interpret meaning.
- As usual, ambiguity is a huge problem.
  - For correctness: need to find the right structure to get the right meaning.
  - For efficiency: searching all possible structures can be very slow; want to use parsing for large-scale language tasks (e.g., used to create Google's "infoboxes").

Global and local ambiguity

- We've already seen examples of **global ambiguity**: multiple analyses for a full sentence, such as I saw the man with the telescope

- But **local ambiguity** is also a big problem: multiple analyses for parts of sentence.
  - the dog bit the child: first three words could be NP (but aren't).
  - Building useless partial structures wastes time.
  - Avoiding useless computation is a major issue in parsing.

- Syntactic ambiguity is rampant; humans usually don't even notice because we are good at using context/semantics to disambiguate.

Parser properties

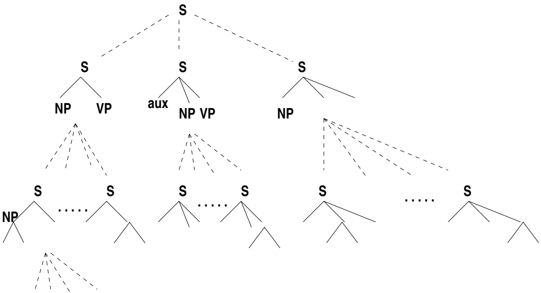
All parsers have two fundamental properties:

- **Directionality**: the sequence in which the structures are constructed.
  - **top-down**: start with root category (S), choose expansions, build down to words.
  - **bottom-up**: build subtrees over words, build up to S.
  - **Mixed** strategies also possible (e.g., left corner parsers)

- **Search strategy**: the order in which the search space of possible analyses is explored.

Example: search space for top-down parser

- Start with S node.
- Choose one of many possible expansions.
- Each of which has children with many possible expansions...
- etc



Search strategies

- **depth-first search**: explore one branch of the search space at a time, as far as possible. If this branch is a dead-end, parser needs to **backtrack.**
- **breadth-first search**: expand all possible branches in parallel (or simulated parallel). Requires storing many incomplete parses in memory at once.
- **best-first search**: score each partial parse and pursue the highest-scoring options first. (Will get back to this when discussing statistical parsing.)

Recursive Descent Parsing

- A **recursive descent** parser treats a grammar as a specification of how to break down a top-level goal (find `S`) into subgoals (find `NP VP`).

- It is a **top-down**, **depth-first** parser:
  - Blindly expand nonterminals until reaching a terminal (word).
  - If multiple options available, choose one but store current state as a backtrack point (in a **stack** to ensure depth-first.)
  - If terminal matches next input word, continue; else, backtrack.

RD Parsing algorithm

Start with subgoal = `S`, then repeat until input/subgoals are empty:

- If first subgoal in list is a **non-terminal** `A`, then pick an expansion `A` $\rightarrow$ `B C` from grammar and replace `A` in subgoal list with `B C`
- If first subgoal in list is a **terminal** `w`:
  - If input is empty, backtrack.
  - If next input word is different from `w`, backtrack.
  - If next input word is `w`, match! i.e., consume input word `w` and subgoal `w` and move to next subgoal.

If we run out of backtrack points but not input, no parse is possible.

Recursive descent example

Consider a very simple example:

- Grammar contains only these rules:

$\texttt{S} \rightarrow \texttt{NP VP}$	$\texttt{VP} \rightarrow \texttt{V}$	$\texttt{NN} \rightarrow \texttt{bit}$	$\texttt{V} \rightarrow \texttt{bit}$
$\texttt{NP} \rightarrow \texttt{DT NN}$	$\texttt{DT} \rightarrow \texttt{the}$	$\texttt{NN} \rightarrow \texttt{dog}$	$\texttt{V} \rightarrow \texttt{dog}$

- The input sequence is the dog bit

Recursive descent example

- Operations:
  - Expand (E)
  - Match (M)
  - Backtrack to step n (Bn)

Step	Op.	Subgoals	Input
0		`S`	the dog bit
1	E	`NP VP`	the dog bit
2	E	`DT NN VP`	the dog bit
3	E	`the NN VP`	the dog bit
4	M	`NN VP`	dog bit
5	E	`bit VP`	dog bit
6	B4	`NN VP`	dog bit
7	E	`dog VP`	dog bit
8	M	`VP`	bit
9	E	`V`	bit
10	E	`bit`	bit
11	M		

Further notes

- The above sketch is actually a **recognizer**: it tells us whether the sentence has a valid parse, but not what the parse is. For a parser, we'd need more details to store the structure as it is built.

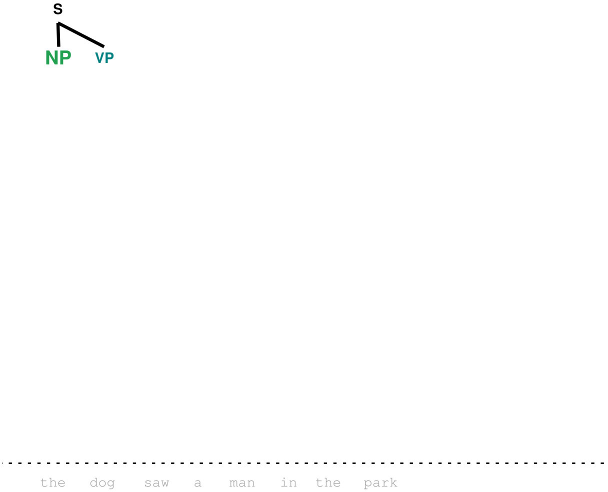
- We only had one backtrack, but in general things can be much worse!
  - If we have left-recursive rules like $\texttt{NP} \rightarrow \texttt{NP PP}$, we get an infinite loop!

Recursive descent parsing example

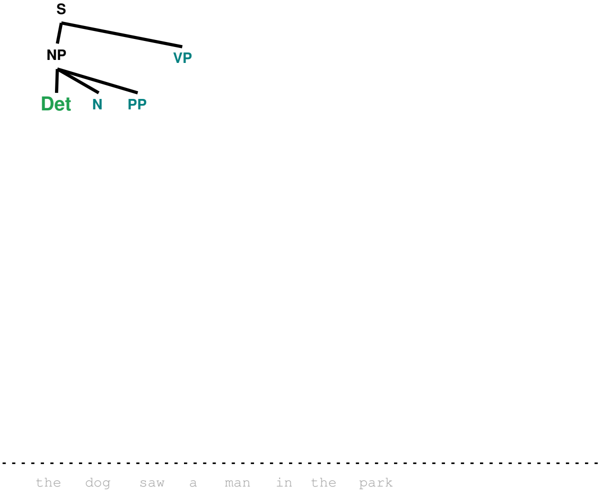
S

the dog saw a man in the park

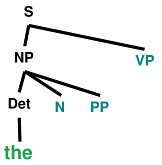
Recursive descent parsing example



Recursive descent parsing example



Recursive descent parsing example

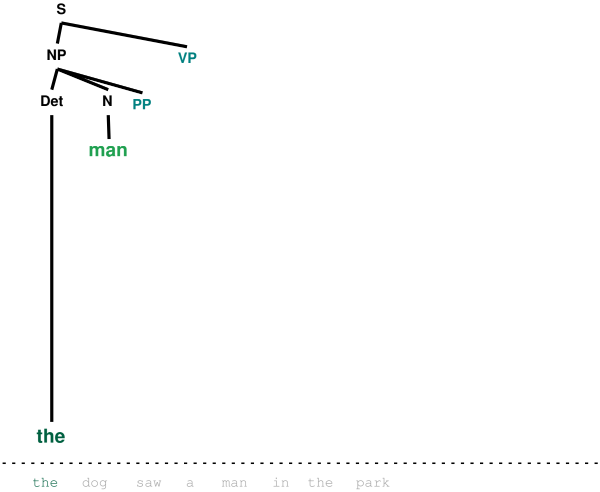


the dog saw a man in the park

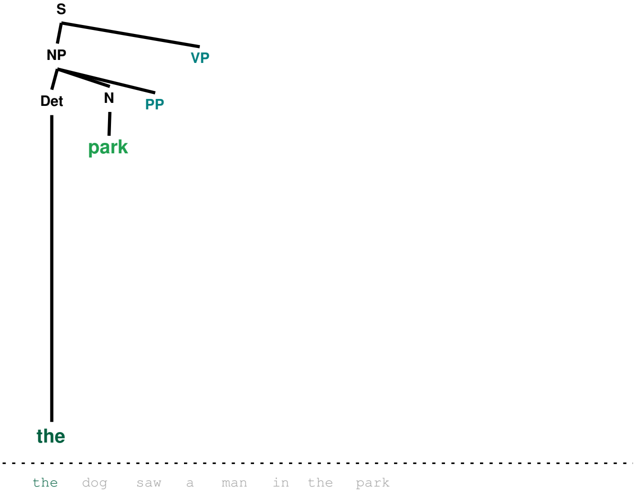
Recursive descent parsing example

the dog saw a man in the park

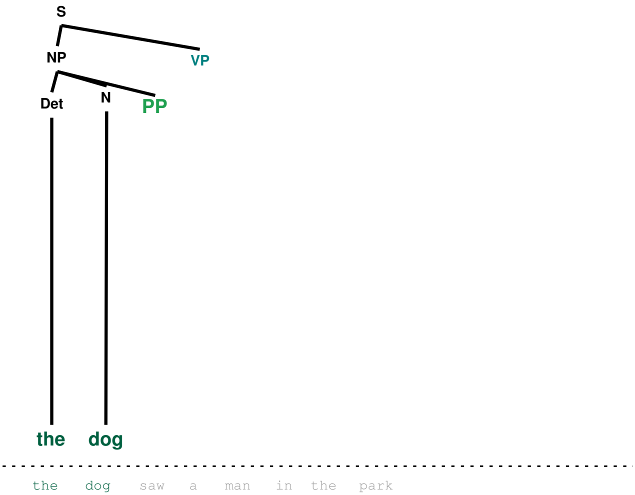
Recursive descent parsing example



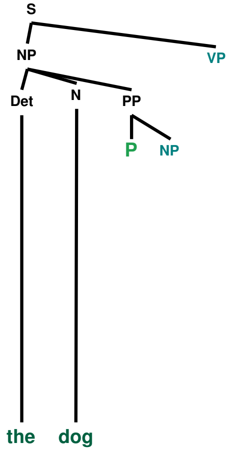
Recursive descent parsing example



Recursive descent parsing example

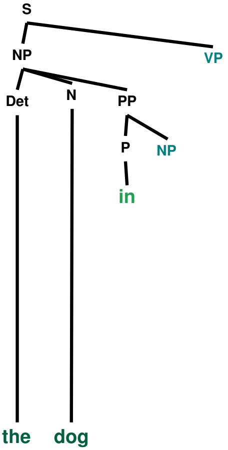


Recursive descent parsing example



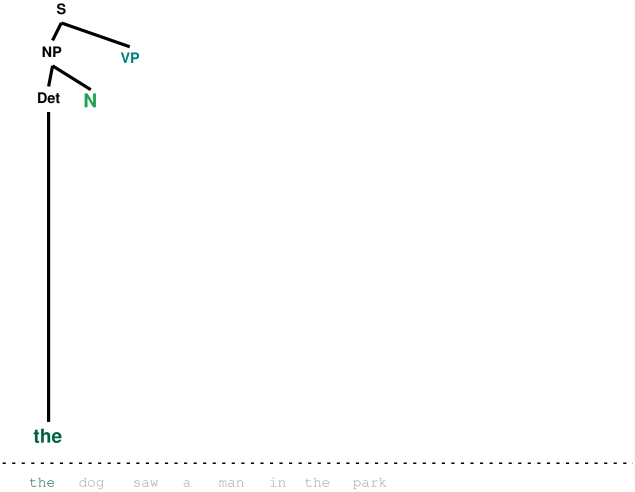
the dog saw a man in the park

Recursive descent parsing example

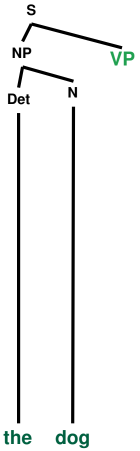


the dog saw a man in the park

Recursive descent parsing example

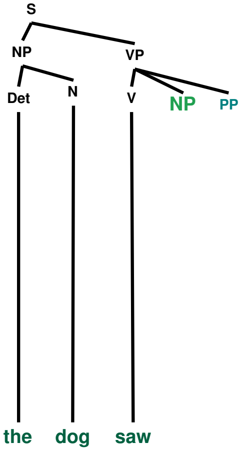


Recursive descent parsing example



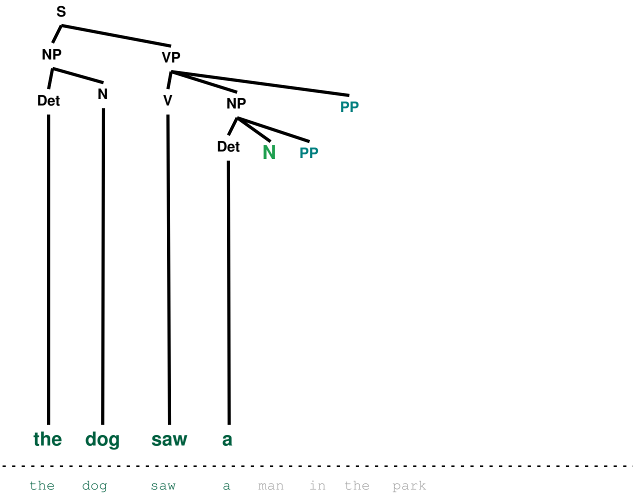
the dog saw a man in the park

Recursive descent parsing example

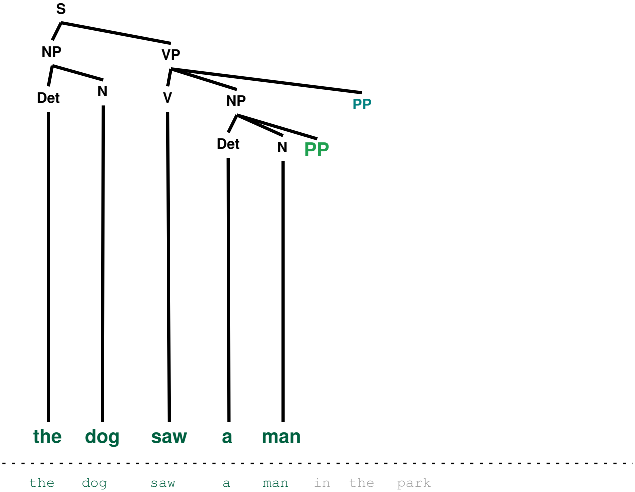


the dog saw a man in the park

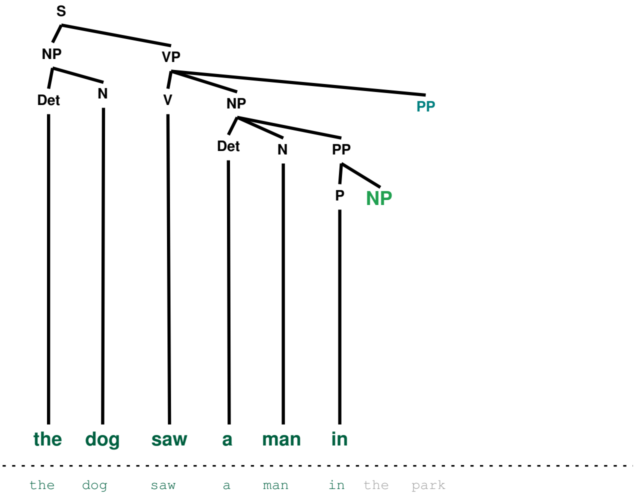
Recursive descent parsing example



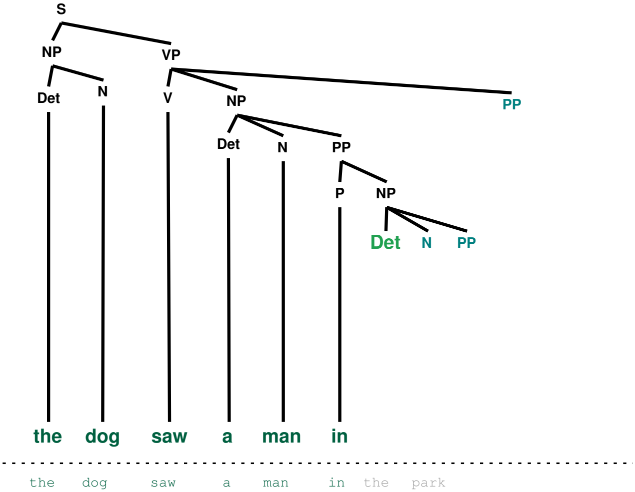
Recursive descent parsing example



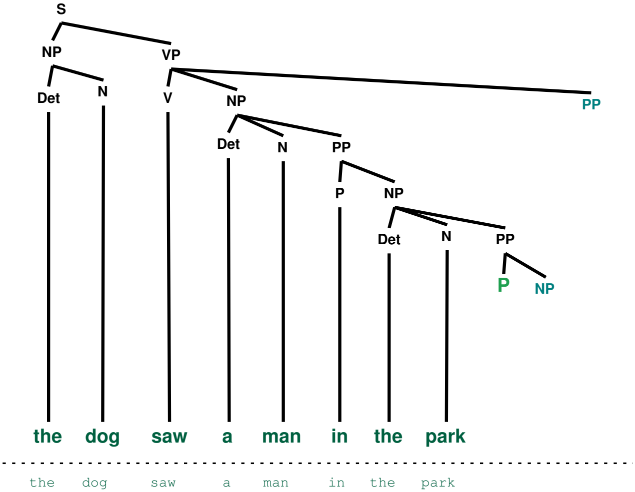
Recursive descent parsing example



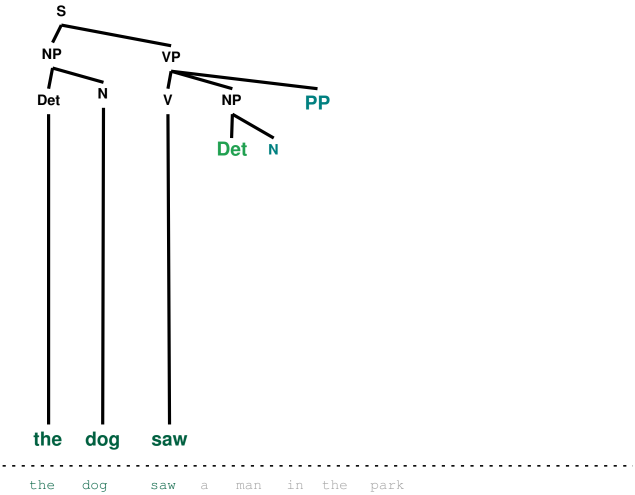
Recursive descent parsing example



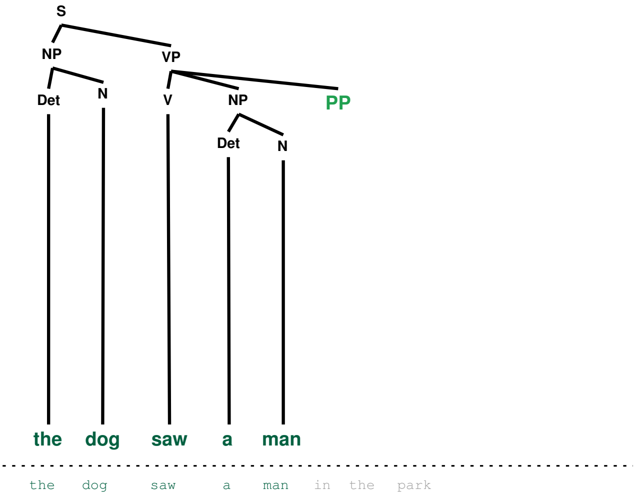
Recursive descent parsing example



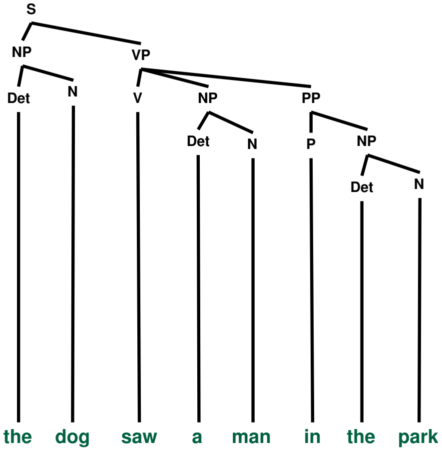
Recursive descent parsing example



Recursive descent parsing example



Recursive descent parsing example



the dog saw a man in the park

Shift-Reduce Parsing

- Search strategy and directionality are orthogonal properties.
- **Shift-reduce** parsing is **depth-first** (like RD) but **bottom-up** (unlike RD).
- Basic shift-reduce recognizer repeatedly:
  - Whenever possible, **reduces** one or more items from top of stack that match RHS of rule, replacing with LHS of rule.
  - When that's not possible, **shifts** an input symbol onto a stack.
- Like RD parser, needs to maintain backtrack points.

Shift-reduce example

- Same example grammar and sentence.
- Operations:
  - Reduce (R)
  - Shift (S)
  - Backtrack to step n (Bn)
- Note that at 9 and 11 we skipped over backtracking to 7 and 5 respectively as there were actually no choices to be made at those points.

Step	Op.	Stack	Input
0			the dog bit
1	S	the	dog bit
2	R	DT	dog bit
3	S	DT dog	bit
4	R	DT V	bit
5	R	DT VP	bit
6	S	DT VP bit	
7	R	DT VP V	
8	R	DT VP VP	
9	B6	DT VP bit	
10	R	DT VP NN	
11	B4	DT V	bit
12	S	DT V bit	
13	R	DT V V	
14	R	DT V VP	
15	B3	DT dog	bit
16	R	DT NN	bit
17	R	NP	bit
...			

Recursive descent parsing example

Stack	Remaining
	my dog saw a man in the park

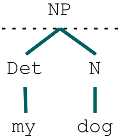
Recursive descent parsing example

Stack	Remaining
Det	dog saw a man in the park

Recursive descent parsing example

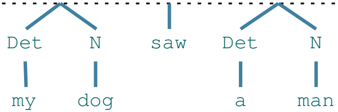
Stack	Remaining
Det N	saw a man in the park
my dog	

Recursive descent parsing example

Stack	Remaining
	saw a man in the park

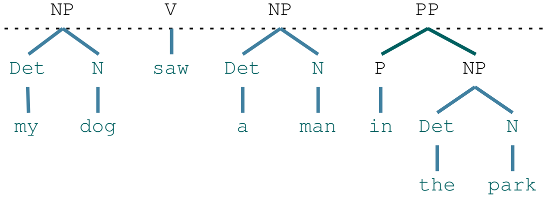
Recursive descent parsing example

Stack	Remaining
NP V NP	in the park

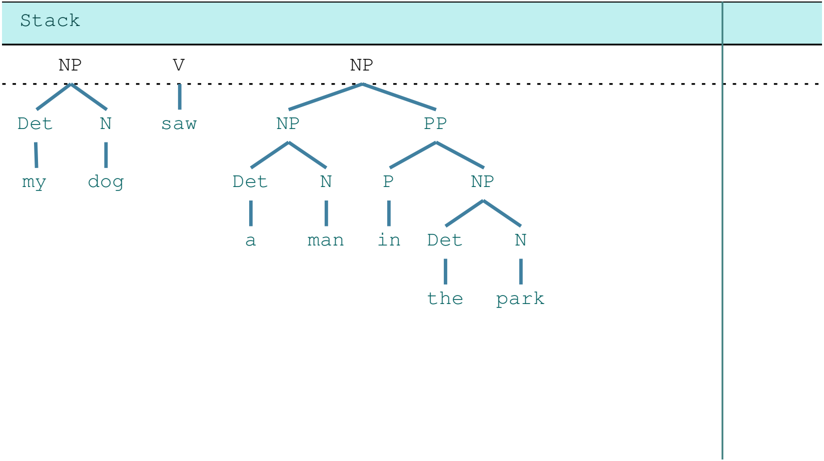


Recursive descent parsing example

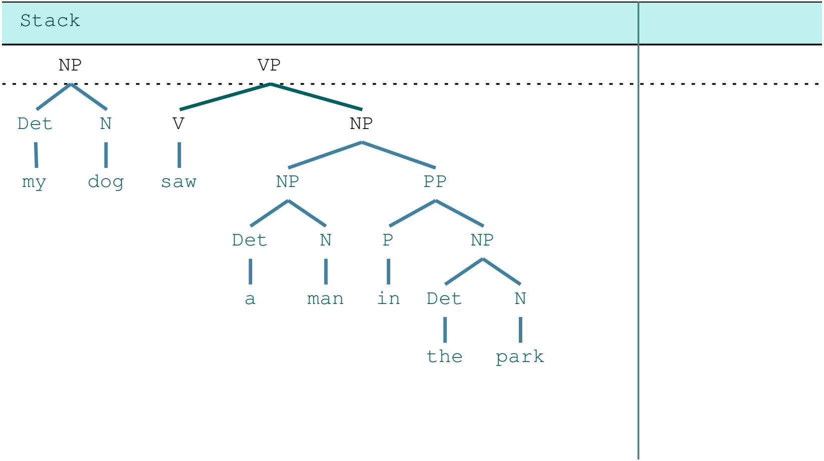
Stack	Remaining



Recursive descent parsing example

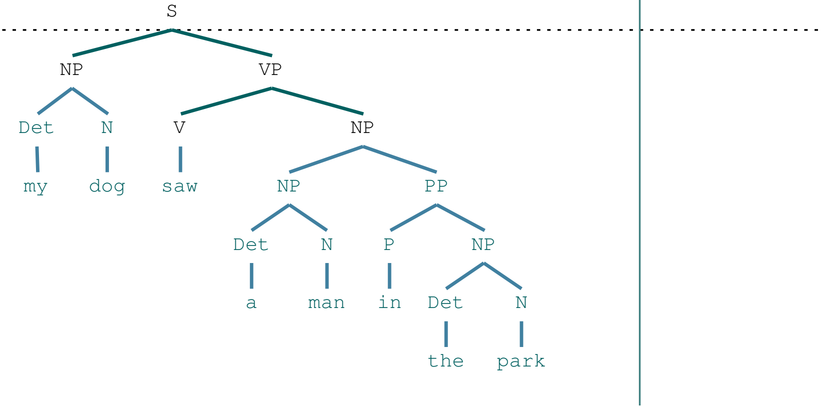


Recursive descent parsing example



Recursive descent parsing example

Stack	



RD and SR parsers in NLTK

Recursive Descent Parser

```
>>> from nltk.app import rdparser
>>> rdparser()
```

Shift-Reduce Parser

```
>>> from nltk.app import srparser
>>> srparser()
```

Depth-first parsing in practice

- Depth-first parsers are very efficient for unambiguous structures.
  - Widely used to parse/compile programming languages, which are constructed to be unambiguous.
- But can be massively inefficient (exponential in sentence length) if faced with local ambiguity.
  - Blind backtracking may require re-building the same structure over and over: so, simple depth-first parsers are not used in NLP.
  - But: if we use a probabilistic model to **learn** which choices to make, we can do very well in practice

Breadth-first search using dynamic programming

- With a CFG, you should be able to avoid re-analysing any substring because its analysis is **independent** of the rest of the parse.

[he]$_{\mathrm{np}}$ [saw her duck]$_{\mathrm{vp}}$

- **chart parsing** algorithms exploit this fact.
  - use dynamic programming to store and reuse sub-parses, composing them into a full solution.
  - So multiple potential parses are explored at once: a breadth-first strategy.

Parsing as dynamic programming

- For parsing, subproblems are analyses of substrings, memoized in **chart** (aka **well-formed substring table**).

- Chart entries are indexed by *start* and *end* positions in the sentence, and correspond to:
  - either a complete **constituent** (sub-tree) spanning those positions (if working bottom-up),
  - **or** a **prediction** about what complete constituent might be found (if working top-down).

What's in the chart?

- We assume **indices** between each word in the sentence:

  $_0$ he $_1$ saw $_2$ her $_3$ duck $_4$

- The chart is a matrix where cell $[i, j]$ holds information about the word span from position i to position j:
  - The root node of any constituent(s) spanning those words
  - Pointers to its sub-constituents
  - (Depending on parsing method,) predictions about what constituents might follow the substring.

Algorithms for Chart Parsing

Many different chart parsing algorithms, including

- the **CKY algorithm**, which memoizes only complete constituents
- various algorithms that also memoize predictions/partial constituents
  - often using mixed bottom-up and top-down approaches, e.g., the Earley algorithm, and left-corner parsing.

CFG Parsing: The Cocke Younger Kasami Algorithm

- Grammar has to be in Chomsky Normal Form (CNF), only
  - RHS with a single terminal: $A \rightarrow a$
  - RHS with two non-terminals: $A \rightarrow BC$
  - no ϵ rules ($A \rightarrow \epsilon$)

- A representation of the string showing positions and word indices:

$$\cdot_0\ w_1\ \cdot_1\ w_2\ \cdot_2\ w_3\ \cdot_3\ w_4\ \cdot_4\ w_5\ \cdot_5\ w_6\ \cdot_6$$

For example: $\cdot_0$ the $\cdot_1$ young $\cdot_2$ boy $\cdot_3$ saw $\cdot_4$ the $\cdot_5$ dragon $\cdot_6$

The well-formed substring table (= passive chart)

- The well-formed substring table, henceforth (passive) chart, for a string of length n is an $n \times n$ matrix.

- The field (i, j) of the chart encodes the set of all categories of constituents that start at position i and end at position j, i.e.

  – chart(i,j) $= \{A \mid A \Rightarrow^* \ w_{i+1} \dots w_j\}$

- The matrix is triangular since no constituent ends before it starts.

Coverage Represented in the Chart

An input sentence with 6 words:

$$\cdot_0\ w_1\ \cdot_1\ w_2\ \cdot_2\ w_3\ \cdot_3\ w_4\ \cdot_4\ w_5\ \cdot_5\ w_6\ \cdot_6$$

Coverage represented in the chart:

TO:

FROM:	1	2	3	4	5	6
0	0–1	0–2	0–3	0–4	0–5	0–6
1		1–2	1–3	1–4	1–5	1–6
2			2–3	2–4	2–5	2–6
3				3–4	3–5	3–6
4					4–5	4–6
5						5–6

Example for Coverage Represented in Chart

Example sentence:

$$\cdot_0 \text{ the } \cdot_1 \text{ young } \cdot_2 \text{ boy } \cdot_3 \text{ saw } \cdot_4 \text{ the } \cdot_5 \text{ dragon } \cdot_6$$

Coverage represented in chart:

	1	2	3	4	5	6
0	the	the young	the young boy	the young boy saw	the young boy saw the	the young boy saw the dragon
1		young	young boy	young boy saw	young boy saw the	young boy saw the dragon
2			boy	boy saw	boy saw the	boy saw the dragon
3				saw	saw the	saw the dragon
4					the	the dragon
5						dragon

Parsing with a Passive Chart

- The CKY algorithm is used, which:
  - explores all analyses in parallel,
  - in a bottom-up fashion, &
  - stores complete subresults
- The reason this algorithm is used is to:
  - add top-down guidance (to only use rules derivable from start-symbol), but avoid left-recursion problem of top-down parsing
  - store partial analyses

An Example for a Filled-in Chart

Input sentence:

$\cdot_0$ the $\cdot_1$ young $\cdot_2$ boy $\cdot_3$ saw $\cdot_4$ the $\cdot_5$ dragon $\cdot_6$

Chart:

	1	2	3	4	5	6
0	{Det}	{}	{NP}	{}	{}	{S}
1		{Adj}	{N}	{}	{}	{}
2			{N}	{}	{}	{}
3				{V, N}	{}	{VP}
4					{Det}	{NP}
5						{N}

Filling in the Chart

- We build all constituents that end at a certain point before we build constituents that end at a later point.

	1	2	3	4	5	6
0	**1**	3	6	10	15	21
1		**2**	5	9	14	20
2			**4**	8	13	19
3				**7**	12	18
4					**11**	17
5						**16**

for j := 1 to length($string$)
    lexical_chart_fill$(j-1, j)$
    for $i := j-2$ down to 0
        syntactic_chart_fill(i, j)

lexical_chart_fill(j-1,j)

- Idea: Lexical lookup. Fill the field $(j-1, j)$ in the chart with the preterminal category dominating word j.
- Realized as:

$$chart(j-1, j) := \{\mathsf{X} \mid \mathsf{X} \rightarrow \mathsf{word}_j \in \mathsf{P}\}$$

syntactic_chart_fill(i,j)

- Idea: Perform all reduction steps using syntactic rules such that the reduced symbol covers the string from i to j.

- Realized as: $chart(i,j) = \left\{ A \;\middle|\; \begin{array}{l} A \rightarrow BC \in P, \\ i < k < j, \\ B \in chart(i,k), \\ C \in chart(k,j) \end{array} \right\}$

- Explicit loops over every possible value of k and every context free rule:

  $chart(i,j) := \{\}$.
  for $k := i + 1$ to $j - 1$
      for every $A \rightarrow BC \in P$
          if $B \in chart(i,k)$ and $C \in chart(k,j)$ then
              $chart(i,j) := chart(i,j) \cup \{\mathsf{A}\}$.

The Complete CYK Algorithm

Input: start category S and input $string$

$n :=$ length($string$)

for $j := 1$ to n
  $chart(j-1, j) := \{\mathsf{X} \mid \mathsf{X} \rightarrow \mathsf{word}_j \in \mathsf{P}\}$
  for $i := j-2$ down to 0
    $chart(i, j) := \{\}$
    for $k := i+1$ to $j-1$
      for every $A \rightarrow BC \in P$
        if $B \in chart(i, k)$ and $C \in chart(k, j)$ then
          $chart(i, j) := chart(i, j) \cup \{\mathsf{A}\}$

Output: if $S \in chart(0, n)$ then accept else reject

How memoization helps

If we look back to the chart for the sentence *the young boy saw the dragon*:

	1	2	3	4	5	6
0	{Det}	{}	{NP}	{}	{}	{S}
1		{Adj}	{N}	{}	{}	{}
2			{N}	{}	{}	{}
3				{V, N}	{}	{VP}
4					{Det}	{NP}
5						{N}

- At cell (3,6), a VP is built by combining the V at (3,4) with the NP at (4,6), based on the rule VP $\rightarrow$ V NP
- Regardless of further processing, that VP is never rebuilt

From recognition to parsing

Extend chart to store in each field

- mother symbol (as before)
- daughters and their field numbers (i.e., backpointers to the structure)

Chart for recovering parses

	1	2	3	4	5	6
0	Det		NP (D,0,1) (N,1,3)			S (NP,0,3) (VP,3,6)
1		Adj	N (A,1,2) (N,2,3)			
2			N			
3				V, N		VP (V,3,4) (NP,4,6)
4					Det	NP (D,4,5) (N,5,6)
5						N

How big a problem is disambiguation?

- Early work in computational linguistics tried to develop broad-coverage hand-written grammars.
  - That is, grammars that *include* all sentences humans would judge as grammatical in their language;
  - while *excluding* all other sentences.

- As coverage grows, sentences can have hundreds or thousands of parses. Very difficult to write heuristic rules for disambiguation.

- Plus, grammar is hard to keep track of! Trying to fix one problem can introduce others.

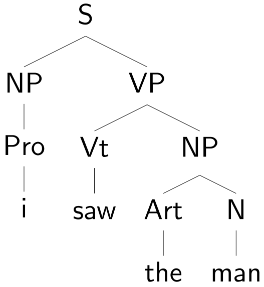
- Enter the **treebank grammar**.

Treebank grammars

- The big idea: instead of paying linguists to write a grammar, pay them to annotate real sentences with parse trees.
- This way, we implicitly get a grammar (for CFG: read the rules off the trees).
- **And** we get probabilities for those rules (using any of our favorite estimation techniques).
- We can use these probabilities to improve disambiguation and even speed up parsing.

Treebank grammars

For example, if we have this tree in our corpus:



Then we add rules

`S` $\rightarrow$ `NP VP`

`NP` $\rightarrow$ `Pro`

`Pro` $\rightarrow$ `i`

`VP` $\rightarrow$ `Vt NP`

`Vt` $\rightarrow$ `saw`

`NP` $\rightarrow$ `Art N`

`Art` $\rightarrow$ `the`

`N` $\rightarrow$ `man`

With more trees, we can start to count rules and estimate their probabilities.

Example: The Penn Treebank

- The first large-scale parse annotation project, begun in 1989.
- Original corpus of syntactic parses: Wall Street Journal text
  - About 40,000 annotated sentences (1m words)
  - Standard phrasal categories like `S`, `NP`, `VP`, `PP`.
- Now many other data sets (e.g. transcribed speech), and different kinds of annotation; also inspired treebanks in many other languages.

Creating a treebank PCFG

A **probabilistic context-free grammar** (PCFG) is a CFG where each rule $\mathtt{A} \to \alpha$ (where α is a symbol sequence) is assigned a probability $P(\alpha|A)$.

- The sum over all expansions of $\mathtt{A}$ must equal 1: $\sum_{\alpha'} P(\alpha'|A) = 1$.
- Easiest way to create a PCFG from a treebank: MLE
  - Count all occurrences of $\mathtt{A} \to \alpha$ in treebank.
  - Divide by the count of all rules whose LHS is $\mathtt{A}$ to get $P(\alpha|A)$
- But as usual many rules have very low frequencies, so MLE isn't good enough and we need to smooth.

The generative model

Like n-gram models and HMMs, PCFGs are a **generative model**. Assumes sentences are generated as follows:

- Start with root category S.
- Choose an expansion α for S with probability $P(\alpha|S)$.
- Then recurse on each symbol in α.
- Continue until all symbols are terminals (nothing left to expand).

The probability of a parse

- Under this model, the probability of a parse t is simply the product of all rules in the parse:

$$P(t) = \prod_{A \to \alpha \in t} A \to \alpha$$

Statistical disambiguation example

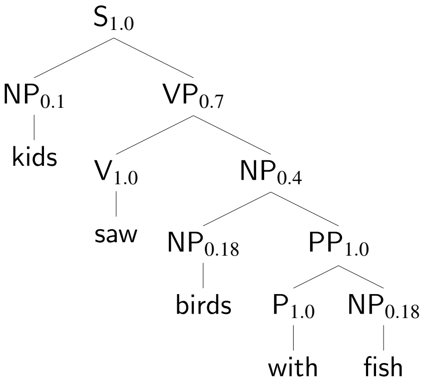
How can parse probabilities help disambiguate PP attachment?

- Let's use the following PCFG, inspired by Manning & Schuetze (1999):

S $\rightarrow$ NP VP	1.0	NP $\rightarrow$ NP PP	0.4
PP $\rightarrow$ P NP	1.0	NP $\rightarrow$ kids	0.1
VP $\rightarrow$ V NP	0.7	NP $\rightarrow$ birds	0.18
VP $\rightarrow$ VP PP	0.3	NP $\rightarrow$ saw	0.04
P $\rightarrow$ with	1.0	NP $\rightarrow$ fish	0.18
V $\rightarrow$ saw	1.0	NP $\rightarrow$ binoculars	0.1

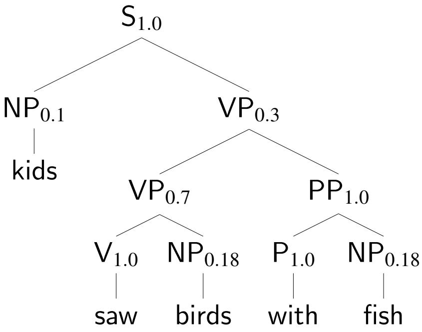
- We want to parse kids saw birds with fish.

Probability of parse 1



- $P(t_1) = 1.0 \cdot 0.1 \cdot 0.7 \cdot 1.0 \cdot 0.4 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0009072$

Probability of parse 2



- $P(t_2) = 1.0 \cdot 0.1 \cdot 0.3 \cdot 0.7 \cdot 1.0 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0006804$
- which is less than $P(t_1) = 0.0009072$, so t_1 is preferred. Yay!

The probability of a sentence

- Since t implicitly includes the words $\vec{w}$, we have $P(t) = P(t, \vec{w})$.

- So, we also have a **language model**. Sentence probability is obtained by summing over $T(\vec{w})$, the set of valid parses of $\vec{w}$:

$$P(\vec{w}) = \sum_{t \in T(\vec{w})} P(t, \vec{w}) = \sum_{t \in T(\vec{w})} P(t)$$

- In our example,
  $P(\text{kids saw birds with fish}) = 0.0006804 + 0.0009072$.

Probabilistic CKY

It is straightforward to extend CKY parsing to the probabilistic case.

- Goal: return the highest probability parse of the sentence (analogous to Viterbi)
  - When we find an `A` spanning (i, j), store its probability along with its label in cell (i, j).
  - If we later find an `A` with the same span but higher probability, replace the probability for `A` in cell (i, j).

	time **1**	flies **2**	like **3**	an **4**	arrow **5**
0	**NP 3** **Vst 3**				
1		**NP 4** **VP 4**			
2			**P 2** **V 5**		
3				**Det 1**	
4					**N 8**

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**				
1		**NP 4** **VP 4**			
2			**P 2** **V 5**		
3				**Det 1**	
4					**N 8**

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10			
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8**			
1		**NP 4** **VP 4**			
2			**P 2** **V 5**		
3				**Det 1**	
4					**N 8**

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			
1		**NP 4** **VP 4**			
2			**P 2** **V 5**		
3				**Det 1**	
4					**N 8**

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time **1** flies **2** like **3** an **4** arrow **5**

0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			
1		**NP 4** **VP 4**			
2			**P 2** **V 5**		
3				**Det 1**	**NP 10**
4					**N 8**

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			
1		**NP 4** **VP 4**			
2			**P 2** **V 5**		**PP 12**
3				**Det 1**	**NP 10**
4					**N 8**

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			
1		**NP 4** **VP 4**			
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time 1 flies 2 like 3 an 4 arrow 5

	1	2	3	4	5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			NP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP

1 VP → V NP

2 VP → VP PP

1 NP → Det N

2 NP → NP PP

3 NP → NP NP

0 PP → P NP

time 1 flies 2 like 3 an 4 arrow 5

	1	2	3	4	5
0	NP 3 Vst 3	NP 10 S 8 S 13			
1		NP 4 VP 4			NP 18 S 21
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

time **1** flies **2** like **3** an **4** arrow **5**

	1	2	3	4	5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			**NP 24** **S 22**
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time 1 flies 2 like 3 an 4 arrow 5

0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			**NP 24** **S 22** **S 27**
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**

1 $S \rightarrow NP\ VP$
6 $S \rightarrow Vst\ NP$
2 $S \rightarrow S\ PP$
1 $VP \rightarrow V\ NP$
2 $VP \rightarrow VP\ PP$
1 $NP \rightarrow Det\ N$
2 $NP \rightarrow NP\ PP$
3 $NP \rightarrow NP\ NP$
0 $PP \rightarrow P\ NP$

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

Follow backpointers …

S

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			**NP 24** **S 22** **S 27** **NP 24** **S 27** **S 22** **S 27**
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**

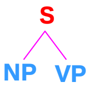
1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP

1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP

1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP

0 PP $\rightarrow$ P NP

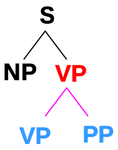
	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			**NP 24** **S 22** **S 27** **NP 24** **S 27** **S 22** **S 27**
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**



1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

time 1 flies 2 like 3 an 4 arrow 5

	1	2	3	4	5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8



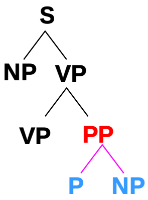
1 S → NP VP
6 S → Vst NP
2 S → S PP

1 VP → V NP
2 VP → VP PP

1 NP → Det N
2 NP → NP PP
3 NP → NP NP

0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8



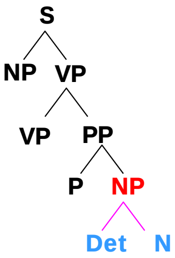
1 S → NP VP
6 S → Vst NP
2 S → S PP

1 VP → V NP
2 VP → VP PP

1 NP → Det N
2 NP → NP PP
3 NP → NP NP

0 PP → P NP

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			**NP 24** **S 22** **S 27** **NP 24** **S 27** **S 22** **S 27**
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**



1 S → NP VP
6 S → Vst NP
2 S → S PP

1 VP → V NP
2 VP → VP PP

1 NP → Det N
2 NP → NP PP
3 NP → NP NP

0 PP → P NP

Which entries do we *need*?

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Which entries do we *need*?

	time 1	flies 2	like 3	an 4	arrow 5
0	**NP 3** **Vst 3**	**NP 10** **S 8** **S 13**			**NP 24** **S 22** **S 27** **NP 24** **S 27** **S 22** **S 27**
1		**NP 4** **VP 4**			**NP 18** **S 21** **VP 18**
2			**P 2** **V 5**		**PP 12** **VP 16**
3				**Det 1**	**NP 10**
4					**N 8**

1 S → NP VP
6 S → Vst NP
2 S → S PP

1 VP → V NP
2 VP → VP PP

1 NP → Det N
2 NP → NP PP
3 NP → NP NP

0 PP → P NP

Not worth keeping …

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 $\boxed{\text{S 13}}$			NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP

1 VP → V NP
2 VP → VP PP

1 NP → Det N
2 NP → NP PP
3 NP → NP NP

0 PP → P NP

... since it just breeds worse options

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 $\boxed{\text{S 13}}$			NP 24 S 22 S 27 NP 24 S 27 S 22 $\boxed{\text{S 27}}$
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP

1 VP → V NP
2 VP → VP PP

1 NP → Det N
2 NP → NP PP
3 NP → NP NP

0 PP → P NP

Keep only best-in-class!

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13	inferior stock		NP 24 S 22 S 27 NP 24 S 27 S 22 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S $\rightarrow$ NP VP
6 S $\rightarrow$ Vst NP
2 S $\rightarrow$ S PP
1 VP $\rightarrow$ V NP
2 VP $\rightarrow$ VP PP
1 NP $\rightarrow$ Det N
2 NP $\rightarrow$ NP PP
3 NP $\rightarrow$ NP NP
0 PP $\rightarrow$ P NP

Keep only best-in-class!

(and its backpointers so you can recover best parse)

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8			NP 24 S 22
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10
4					N 8

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Probabilistic CKY example

function Probabilistic-CYK(*words*, *grammar*) **returns** most probable parse and its probability

for j ←**from** 1 **to** $\textsc{Length}(words)$ **do**
  for all $\{A | A \rightarrow words[j] \in grammar\}$
   $table[j-1, j, A] \leftarrow P(A \rightarrow words[j])$
  for i ←**from** $j-2$ **downto** 0 **do**
   for all $\{A | A \rightarrow BC \in grammar,$
       and $table[i,k,B] > 0$ **and** $table[k,j,C] > 0\}$
    if$(table[i,j,A] < P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C])$**then**
     $table[i,j,A] \leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$
     $back[i,j,A] \leftarrow \{k, B, C\}$
return
$\textsc{Build\_Tree}(back[1, \textsc{Length}(words), S]), table[1, \textsc{Length}(words), S]$

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0,1]$				

$S \rightarrow NP\ VP$ .80	$Det \rightarrow the$ .40
$NP \rightarrow Det\ N$ .30	$Det \rightarrow a$ .40
$VP \rightarrow V\ NP$ .20	$N \rightarrow meal$ .01
$V \rightarrow includes$ .05	$N \rightarrow flight$ .02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0,1]$				
	N: .02 $[1,2]$			

$S \rightarrow NP\ VP$ .80	$Det \rightarrow the$ .40
$NP \rightarrow Det\ N$ .30	$Det \rightarrow a$ .40
$VP \rightarrow V\ NP$ .20	$N \rightarrow meal$ .01
$V \rightarrow includes$ .05	$N \rightarrow flight$ .02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 [0, 1]				
	N: .02 [1, 2]			
		V: .05 [2, 3]		

$S \rightarrow NP\ VP$ .80	$Det \rightarrow the$ .40
$NP \rightarrow Det\ N$ .30	$Det \rightarrow a$ .40
$VP \rightarrow V\ NP$ .20	$N \rightarrow meal$ .01
$V \rightarrow includes$ .05	$N \rightarrow flight$ .02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 [0, 1]				
	N: .02 [1, 2]			
		V: .05 [2, 3]		
			Det: .40 [3, 4]	

$S \rightarrow NP\ VP$	.80		$Det \rightarrow the$	.40	
$NP \rightarrow Det\ N$	.30		$Det \rightarrow a$	.40	
$VP \rightarrow V\ NP$	.20		$N \rightarrow meal$	.01	
$V \rightarrow includes$	.05		$N \rightarrow flight$	.02	

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 [0, 1]				
	N: .02 [1, 2]			
		V: .05 [2, 3]		
			Det: .40 [3, 4]	
				N: .01 [4, 5]

$S \rightarrow NP\ VP\ .80$	$Det \rightarrow the .40$
$NP \rightarrow Det\ N .30$	$Det \rightarrow a\ \ .40$
$VP \rightarrow V\ NP .20$	$N \rightarrow meal .01$
$V \rightarrow includes .05$	$N \rightarrow flight .02$

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0,1]$	NP: $.30 \times .40 \times .02 = .0024$ $[0,2]$			
	N: .02 $[1,2]$			
		V: .05 $[2,3]$		
			Det: .40 $[3,4]$	
				N: .01 $[4,5]$

$S \rightarrow NP\ VP\ .80$	$Det \rightarrow the .40$
$NP \rightarrow Det\ N .30$	$Det \rightarrow a\ .40$
$VP \rightarrow V\ NP .20$	$N \rightarrow meal .01$
$V \rightarrow includes .05$	$N \rightarrow flight .02$

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 [0, 1]	NP: $.30 \times .40 \times .02 = .0024$ [0, 2]			
	N: .02 [1, 2]	[1, 3]		
		V: .05 [2, 3]		
			Det: .40 [3, 4]	
				N: .01 [4, 5]

$S \rightarrow NP\ VP\ .80$	$Det \rightarrow the .40$
$NP \rightarrow Det\ N .30$	$Det \rightarrow a\ .40$
$VP \rightarrow V\ NP\ .20$	$N \rightarrow meal\ .01$
$V \rightarrow includes .05$	$N \rightarrow flight .02$

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 [0, 1]	NP: $.30 \times .40 \times .02 = .0024$ [0, 2]	[0, 3]		
	N: .02 [1, 2]	[1, 3]		
		V: .05 [2, 3]		
			Det: .40 [3, 4]	
				N: .01 [4, 5]

$S \rightarrow NP\ VP$ .80	$Det \rightarrow the$ .40
$NP \rightarrow Det\ N$ .30	$Det \rightarrow a$ .40
$VP \rightarrow V\ NP$ .20	$N \rightarrow meal$ .01
$V \rightarrow includes$ .05	$N \rightarrow flight$ .02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0, 1]$	NP: $.30 \times .40 \times .02 = .0024$ $[0, 2]$	$[0, 3]$		
	N: .02 $[1, 2]$	$[1, 3]$		
		V: .05 $[2, 3]$	$[2, 4]$	
			Det: .40 $[3, 4]$	
				N: .01 $[4, 5]$

$S \rightarrow NP\ VP\ .80$	$Det \rightarrow the .40$
$NP \rightarrow Det\ N .30$	$Det \rightarrow a\ .40$
$VP \rightarrow V\ NP .20$	$N \rightarrow meal\ .01$
$V \rightarrow includes .05$	$N \rightarrow flight .02$

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0, 1]$	NP: $.30 \times .40 \times .02 = .0024$ $[0, 2]$	$[0, 3]$		
	N: .02 $[1, 2]$	$[1, 3]$	$[1, 4]$	
		V: .05 $[2, 3]$	$[2, 4]$	
			Det: .40 $[3, 4]$	
				N: .01 $[4, 5]$

$S \rightarrow NP\ VP$	.80	$Det \rightarrow the$	.40
$NP \rightarrow Det\ N$	.30	$Det \rightarrow a$	.40
$VP \rightarrow V\ NP$	.20	$N \rightarrow meal$	.01
$V \rightarrow includes$	.05	$N \rightarrow flight$	.02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 [0, 1]	NP: $.30 \times .40 \times .02 = .0024$ [0, 2]	[0, 3]	[0, 4]	
	N: .02 [1, 2]	[1, 3]	[1, 4]	
		V: .05 [2, 3]	[2, 4]	
			Det: .40 [3, 4]	
				N: .01 [4, 5]

$S \rightarrow NP\ VP$ .80	$Det \rightarrow the$ .40
$NP \rightarrow Det\ N$ .30	$Det \rightarrow a$ .40
$VP \rightarrow V\ NP$ .20	$N \rightarrow meal$ .01
$V \rightarrow includes$ .05	$N \rightarrow flight$ .02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 [0, 1]	NP: $.30 \times .40 \times .02 = .0024$ [0, 2]	[0, 3]	[0, 4]	
	N: .02 [1, 2]	[1, 3]	[1, 4]	
		V: .05 [2, 3]	[2, 4]	
			Det: .40 [3, 4]	NP: $.30 \times .40 \times .01 = 0.0012$ [3, 5]
				N: .01 [4, 5]

$S \to NP\ VP\ .80$	$Det \to the\ .40$
$NP \to Det\ N\ .30$	$Det \to a\ .40$
$VP \to V\ NP\ .20$	$N \to meal\ .01$
$V \to includes\ .05$	$N \to flight\ .02$

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0,1]$	NP: $.30 \times .40 \times .02 = .0024$ $[0,2]$	$[0,3]$	$[0,4]$	
	N: .02 $[1,2]$	$[1,3]$	$[1,4]$	
		V: .05 $[2,3]$	$[2,4]$	VP: $.20 \times .05 \times 0.0012 = 0.000012$ $[2,5]$
			Det: .40 $[3,4]$	NP: $.30 \times .40 \times .01 = 0.0012$ $[3,5]$
				N: .01 $[4,5]$

$S \rightarrow NP\ VP$ .80	$Det \rightarrow the$ .40
$NP \rightarrow Det\ N$ .30	$Det \rightarrow a$ .40
$VP \rightarrow V\ NP$ .20	$N \rightarrow meal$ .01
$V \rightarrow includes$ .05	$N \rightarrow flight$ .02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0,1]$	NP: $.30 \times .40 \times .02 = .0024$ $[0,2]$	$[0,3]$	$[0,4]$	
	N: .02 $[1,2]$	$[1,3]$	$[1,4]$	$[1,5]$
		V: .05 $[2,3]$	$[2,4]$	VP: $.20 \times .05 \times 0.0012 = 0.000012$ $[2,5]$
			Det: .40 $[3,4]$	NP: $.30 \times .40 \times .01 = 0.0012$ $[3,5]$
				N: .01 $[4,5]$

$S \rightarrow NP\ VP$ .80	$Det \rightarrow the$ .40
$NP \rightarrow Det\ N$ .30	$Det \rightarrow a$ .40
$VP \rightarrow V\ NP$ .20	$N \rightarrow meal$ .01
$V \rightarrow includes$ .05	$N \rightarrow flight$ .02

Probabilistic CKY example

The	flight	includes	a	meal
Det: .40 $[0,1]$	NP: $.30 \times .40 \times .02 = .0024$ $[0,2]$	$[0,3]$	$[0,4]$	S: $.80 \times .0024 \times .000012 = .000000023$ $[0,5]$
	N: .02 $[1,2]$	$[1,3]$	$[1,4]$	$[1,5]$
		V: .05 $[2,3]$	$[2,4]$	VP: $.20 \times .05 \times 0.0012 = 0.000012$ $[2,5]$
			Det: .40 $[3,4]$	NP: $.30 \times .40 \times .01 = 0.0012$ $[3,5]$
				N: .01 $[4,5]$

$S \rightarrow NP\ VP\ .80$	$Det \rightarrow the\ .40$
$NP \rightarrow Det\ N\ .30$	$Det \rightarrow a\ .40$
$VP \rightarrow V\ NP\ .20$	$N \rightarrow meal\ .01$
$V \rightarrow includes\ .05$	$N \rightarrow flight\ .02$

Probabilistic CKY example

$$
\begin{array}{rcl}
\mathsf{S} & \rightarrow & \mathsf{NP\ VP\ (1.0)} \\
\mathsf{NP} & \rightarrow & \mathsf{N\ (0.6)} \ \mid\ \mathsf{A\ NP\ (0.2)} \ \mid\ \mathsf{NP\ N\ (0.2)} \\
\mathsf{VP} & \rightarrow & \mathsf{V\ (0.8)} \ \mid\ \mathsf{V\ Adv\ (0.2)} \\
\mathsf{N} & \rightarrow & \mathsf{orange\ (0.3)} \ \mid\ \mathsf{tree\ (0.5)} \ \mid\ \mathsf{blossoms\ (0.2)} \\
\mathsf{A} & \rightarrow & \mathsf{orange\ (1.0)} \\
\mathsf{V} & \rightarrow & \mathsf{blossoms\ (1.0)} \\
\mathsf{Adv} & \rightarrow & \mathsf{early\ (1.0)}
\end{array}
$$

(Not quite in CNF, but never mind.) We'll parse:

orange tree blossoms early

Probabilistic CKY example

	orange	tree	blossoms	early
orange	N (0.3) A (1.0) NP (0.18)	NP (0.06)	S (0.048) NP (0.0024)	S (0.012)
tree		N (0.5) NP (0.3)	NP (0.012)	S (0.06)
blossoms			N (0.2) V (1.0) NP (0.12) VP (0.8)	VP (0.2)
early				Adv(1.0)

Probabilistic CKY example

- The phrase orange tree gets 0.06 for its best analysis *as an NP*, since

0.06	= 0.2*1.0*0.3	(for NP $\rightarrow$ A NP)
beats 0.018	= 0.18*0.5*0.2	(for NP $\rightarrow$ NP N).

  Only the higher probability is recorded in the chart.
- For orange tree blossoms, there are now two analyses as NP, each with probability 0.0024.
- There is also an analysis of orange tree blossoms as S. This doesn't compete with its analysis as NP, so both are recorded.

Best-first probabilistic parsing

- So far, we've been assuming **exhaustive** parsing: return all possible parses.

- But treebank grammars are huge!! Exhaustive parsing of WSJ sentences up to 40 words long adds on average over 1m items to chart per sentence.[1]

- **Best-first** parsing can help.

[1] Charniak, Goldwater, and Johnson, WVLC 1998.

Best-first probabilistic parsing

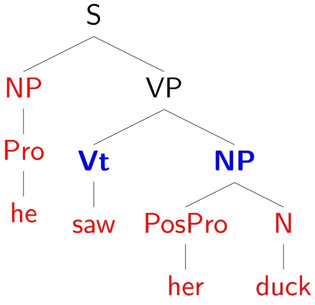
Basic idea: use probabilities of subtrees to decide which ones to build up further.

- Each time we find a new constituent, we give it a **score** ("figure of merit") and add it to an **agenda**, which is ordered by score.
- Then we pop the next item off the agenda, add it to the chart, and see which new constituents we can make using it.
- We add those to the agenda, and iterate.

Notice we are no longer filling the chart in any fixed order.

Best-first intuition

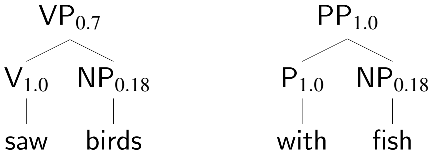
Suppose red constituents are in chart already; **blue** are on agenda.



If the `VP` in right-hand tree scores high enough, we'll pop that next, add it to chart, then find the `S`. So, we could complete the whole parse before even finding the alternative `VP`.

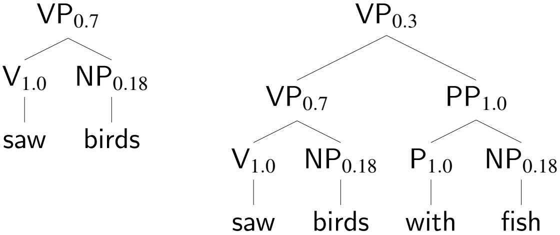
How do we score constituents?

Perhaps according to the probability of the subtree they span? So, P(left example)=(0.7)(0.18) and P(right example)=0.18.



How do we score constituents?

But what about comparing different sized constituents?



A better figure of merit

- If we use raw probabilities for the score, **smaller** constituents will almost always have higher scores.
  - Meaning we pop all the small constituents off the agenda before the larger ones.
  - Which would be very much like exhaustive bottom-up parsing!

- Instead, we can divide by the **number of words** in the constituent.
  - Very much like we did when comparing language models (recall **per-word** cross-entropy)!

- This works much better, though still not guaranteed to find the best parse first. Other improvements are possible (including A*).

But wait a minute...

Best-first parsing shows how simple ("vanilla") treebank PCFGs can improve **efficiency**. But do they really solve the problem of disambiguation?

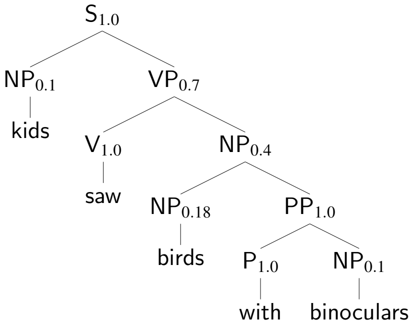
- Our example grammar gave the right parse for this sentence:

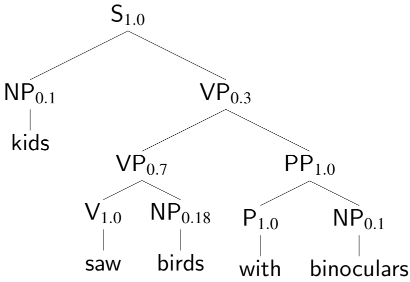
  kids saw birds with fish

- What happens if we parse this sentence?

  kids saw birds with binoculars

Vanilla PCFGs: no lexical dependencies





- Exactly the same probabilities as the "fish" trees, except divide out $P(\text{fish}|\texttt{NP})$ and multiply in $P(\text{binoculars}|\texttt{NP})$ in each case.

- So, the same (left) tree is preferred, but now incorrectly!

Vanilla PCFGs: no lexical dependencies

Replacing one word with another with the same POS will never result in a different parsing decision, even though it should!

- More examples:
  - She stood by the door covered in tears vs.
    She stood by the door covered in ivy
  - She called on the student vs.
    She called on the phone.
    (assuming "on" has the same POS...)

Vanilla PCFGs: no global structural preferences

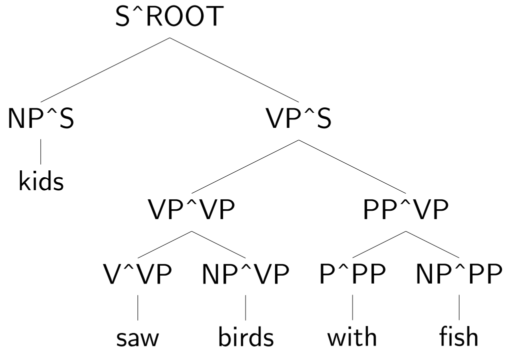
- Ex. in Switchboard corpus, the probability of NP → Pronoun
  - in **subject position** is 0.91 — he saw the dog
  - in **object position** is 0.34 — the dog bit him
- Lots of other rules also have different probabilities depending on where they occur in the sentence.
- But PCFGs are context-free, so an NP is an NP is an NP, and will have the same expansion probs regardless of where it appears.

Ways to fix PCFGs (1): parent annotation

Automatically create new categories that include the old category and its parent.

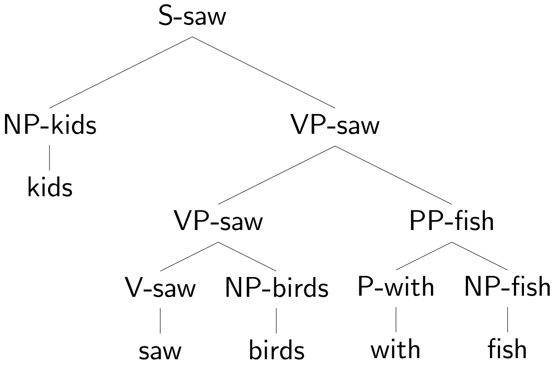
- So, an `NP` in subject position becomes `NP^S`, with other NPs becoming `NP^VP`, `NP^PP`, etc.
- Ex. rules:
  - `S^ROOT` $\rightarrow$ `NP^S VP^S`
  - `NP^S` $\rightarrow$ `Pro^NP`
  - `NP^S` $\rightarrow$ `NP^NP PP^NP`

Example of parent annotation



Ways to fix PCFGs (2): lexicalization

Again, create new categories, this time by adding the **lexical head** of the phrase:



- Now consider:

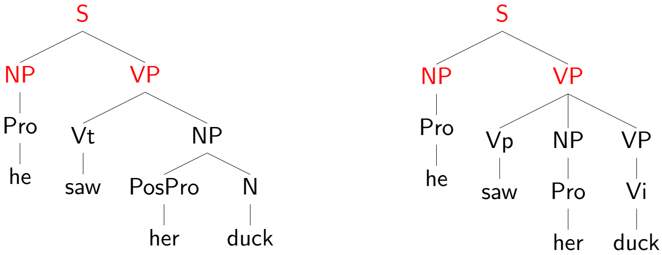
`VP-saw` $\rightarrow$ `VP-saw PP-fish` vs. `VP-saw` $\rightarrow$ `VP-saw PP-binoculars`

Practical issues, again

- All this category-splitting makes the grammar much more **specific** (good!)
- But leads to huge grammar blowup and very sparse data (bad!)
- Lots of effort on how to balance these two issues.
  - Complex smoothing schemes (similar to N-gram interpolation/backoff).
  - More recently, increasing emphasis on automatically learned subcategories.
- But how do we know which method works best?

Evaluating parse accuracy

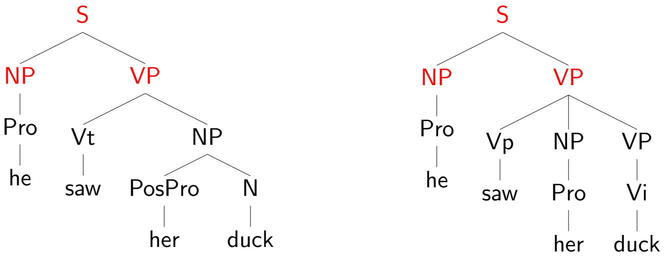
Compare **gold standard** tree (left) to **parser output** (right):



- Output constituent is counted correct if there is a gold constituent that spans the same sentence positions.
- Harsher measure: also require the constituent labels to match.
- Pre-terminals don't count as constituents.

Evaluating parse accuracy

Compare **gold standard** tree (left) to **parser output** (right):



- **Precision**: (# correct constituents)/(# in parser output) = 3/5
- **Recall**: (# correct constituents)/(# in gold standard) = 3/4
- **F-score**: balances precision/recall: 2pr/(p+r)

Parsing accuracies

F-scores for parsing on WSJ corpus:

- vanilla PCFG: $< 80\%$[2]
- lexicalizing $+$ cat-splitting: 89.5% (Charniak, 2000)
- Best current parsers get about 92%

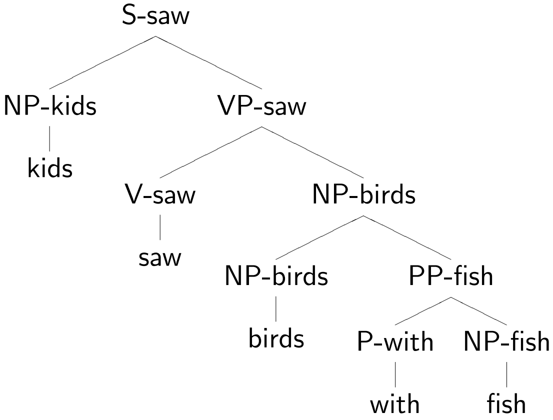
However, results on other corpora and other languages are considerably lower. Definitely not a solved problem!

[2] Charniak (1996) reports 81% but using gold POS tags as input.

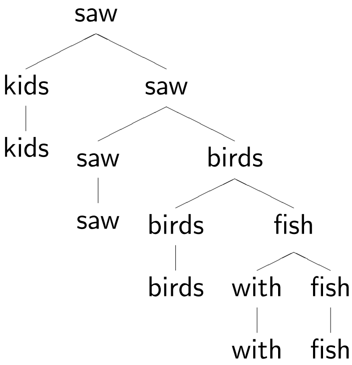
Practical issues, again

- Leads to huge grammar blowup and very sparse data (bad!)
  - There are fancy techniques to address these issues. . .
  - But: Do we really need phrase structures in the first place? Not always!
- Today: Syntax without constituent structure.

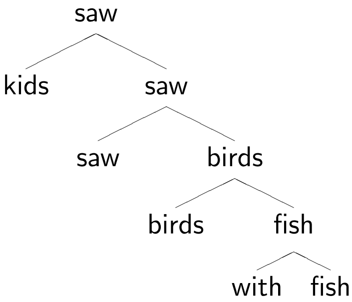
Lexicalized Constituency Parse



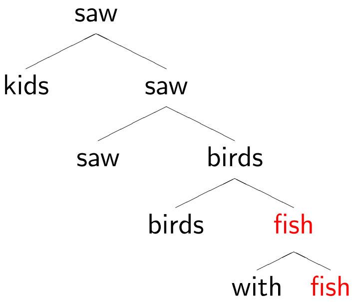
. . . remove the phrasal categories. . .



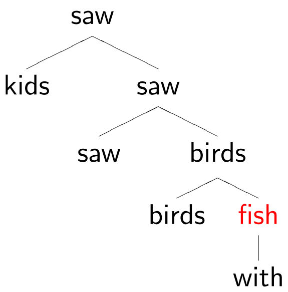
. . . remove the (duplicated) terminals. . .



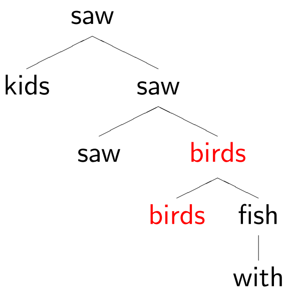
. . . and collapse chains of duplicates. . .



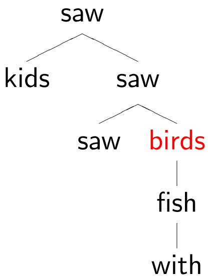
. . . and collapse chains of duplicates. . .



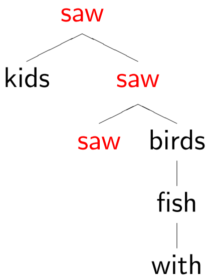
. . . and collapse chains of duplicates. . .



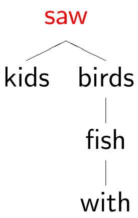
. . . and collapse chains of duplicates. . .



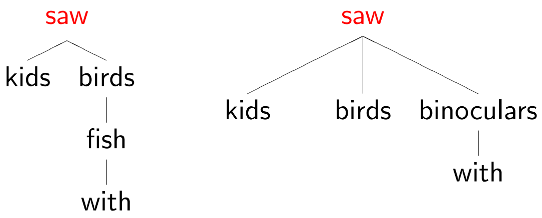
. . . and collapse chains of duplicates. . .



. . . and collapse chains of duplicates. . .



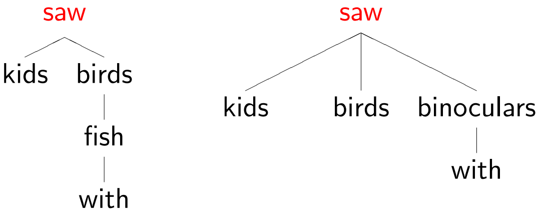
Dependency Parse



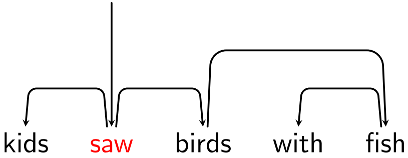
Linguists have long observed that the meanings of words within a sentence depend on one another, mostly in *asymmetric*, *binary* relations.

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

Dependency Parse



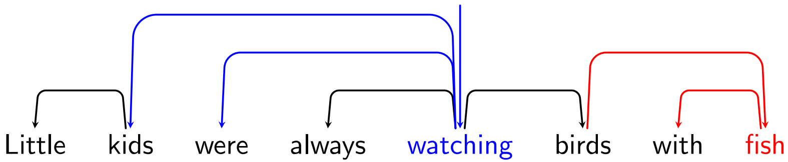
Equivalently, but showing word order (head → modifier):



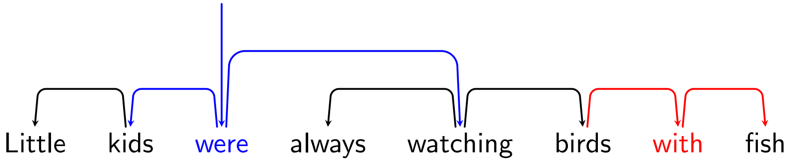
Because it is a tree, every word has exactly one parent.

Content vs. Functional Heads

Some treebanks prefer **content heads**:

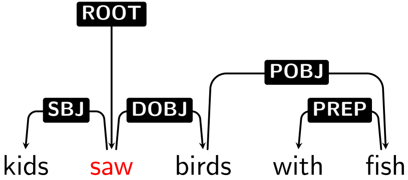


Others prefer **functional heads**:



Edge Labels

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

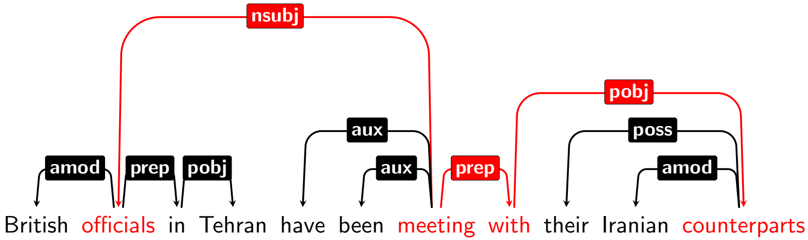


Important relations for English include subject, direct object, determiner, adjective modifier, adverbial modifier, etc. (Different treebanks use somewhat different label sets.)

- How would you identify the subject in a constituency parse?

Dependency Paths

For **information extraction** tasks involving real-world relationships between entities, chains of dependencies can provide good features:

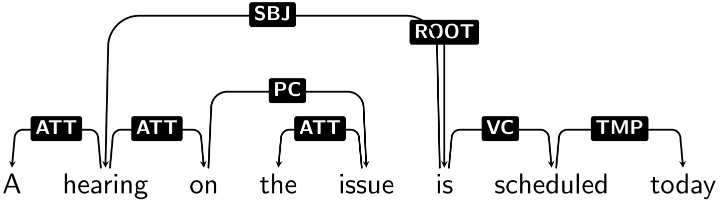


(example from Brendan O'Connor)

Projectivity

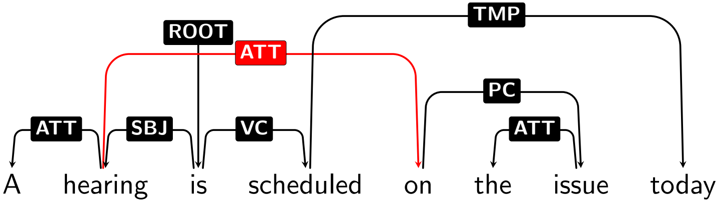
- A sentence's dependency parse is said to be **projective** if every subtree (node and all its descendants) occupies a *contiguous span* of the sentence.

- = The dependency parse can be drawn on top of the sentence without any crossing edges.



Nonprojectivity

- Other sentences are **nonprojective**:



- Nonprojectivity is rare in English, but quite common in many languages.

Dependency Parsing

Some of the algorithms you have seen for PCFGs can be adapted to dependency parsing.

- **CKY** can be adapted, though efficiency is a concern: obvious approach is $O(Gn^5)$; Eisner algorithm brings it down to $O(Gn^3)$
  - N. Smith's slides explaining the Eisner algorithm: `http://courses.cs.washington.edu/courses/cse517/16wi/slides/an-dep-slides.pdf`
- **Shift-reduce**: more efficient, doesn't even require a grammar!

Transition-based Parsing

- Adapts shift-reduce methods: stack and buffer
- Remember: latent structure is just edges between words. Train a **classifier** to predict next action (SHIFT, REDUCE, ATTACH-LEFT, or ATTACH-RIGHT), and proceed left-to-right through the sentence. $O(n)$ time complexity!
- Only finds **projective** trees (without special extensions)
- Pioneering system: Nivre's MALTPARSER
- See `http://spark-public.s3.amazonaws.com/nlp/slides/Parsing-Dependency.pdf` (Jurafsky & Manning Coursera slides) for details and examples

Graph-based Parsing

- Global algorithm: From the fully connected directed graph of all possible edges, choose the best ones that form a tree.

- **Edge-factored** models: Classifier assigns a nonnegative score to each possible edge; **maximum spanning tree** algorithm finds the spanning tree with highest total score in $O(n^2)$ time.
  - Edge-factored assumption can be relaxed (higher-order models score larger units; more expensive).
  - Unlabeled parse $\rightarrow$ edge-labeling classifier (pipeline).

- Pioneering work: McDonald's MSTParser

- Can be formulated as constraint-satisfaction with **integer linear programming** (Martins's TurboParser)

Example transition-based parsing : the arc-standard algorithm

- The arc-standard algorithm is a simple algorithm for transition-based dependency parsing.
- It is very similar to shift–reduce parsing as it is known for context-free grammars.
- It is implemented in most practical transition-based dependency parsers, including MaltParser.

Example transition-based parsing : the arc-standard algorithm

A configuration for a sentence $w = w_1 \ldots w_n$ consists of three components:

- a buffer containing words of w
- a stack containing words of w
- the dependency graph constructed so far

Example transition-based parsing : the arc-standard algorithm

- Initial configuration:
  - All words are in the buffer.
  - The stack is empty.
  - The dependency graph is empty.
- Terminal configuration:
  - The buffer is empty.
  - The stack contains a single word.

Example transition-based parsing : the arc-standard algorithm

- shift (sh): push
  the next word in the buffer onto the stack
- left-arc (la): add an arc
  from the topmost word on the stack, s_1,
  to the second-topmost word, s_2, and pop s_2
- right-arc (ra): add an arc
  from the second-topmost word on the stack, s_2,
  to the topmost word, s_1, and pop s_1

Example transition-based parsing : the arc-standard algorithm

Stack

Buffer

I	booked	a	flight	from LA

I booked a flight from LA

Example transition-based parsing : the arc-standard algorithm

Stack

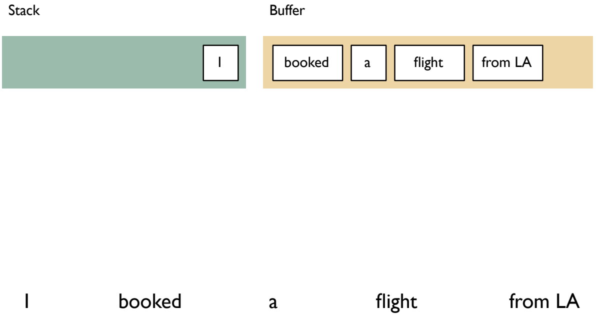
Buffer

I	booked	a	flight	from LA

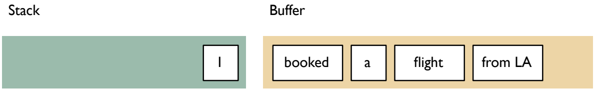
I booked a flight from LA

sh

Example transition-based parsing : the arc-standard algorithm



Example transition-based parsing : the arc-standard algorithm



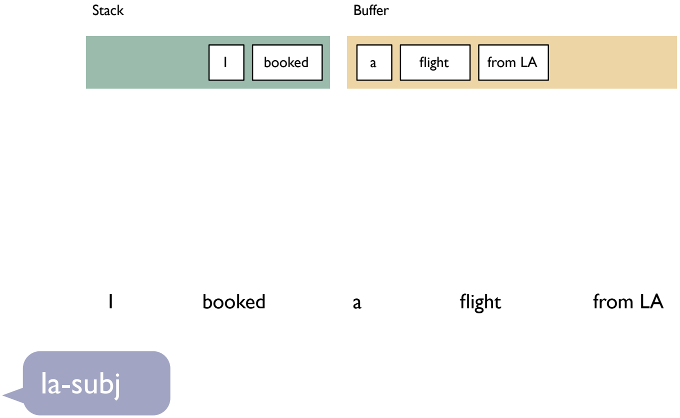
I booked a flight from LA

Example transition-based parsing : the arc-standard algorithm

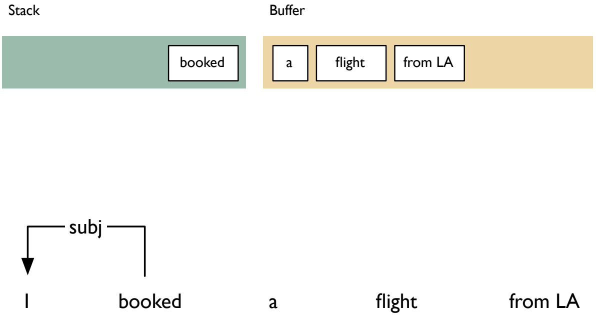
Stack		Buffer		
I	booked	a	flight	from LA

I booked a flight from LA

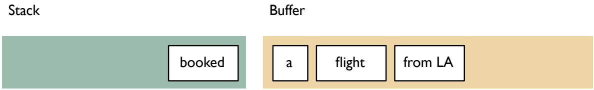
Example transition-based parsing : the arc-standard algorithm

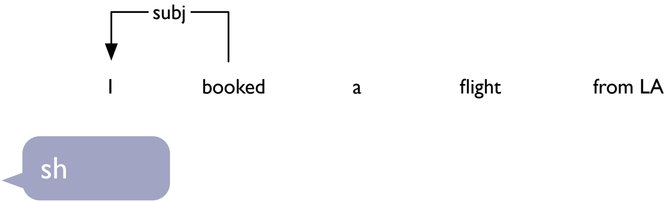


Example transition-based parsing : the arc-standard algorithm

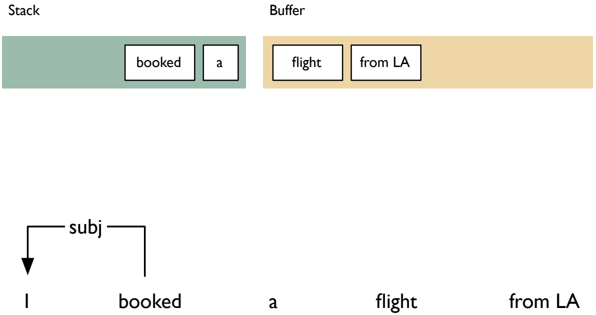


Example transition-based parsing : the arc-standard algorithm

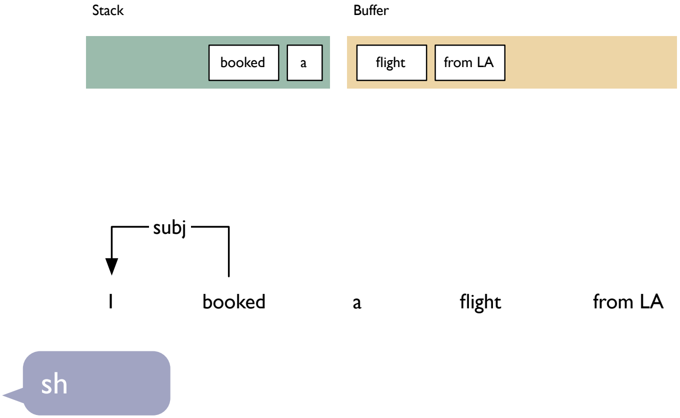




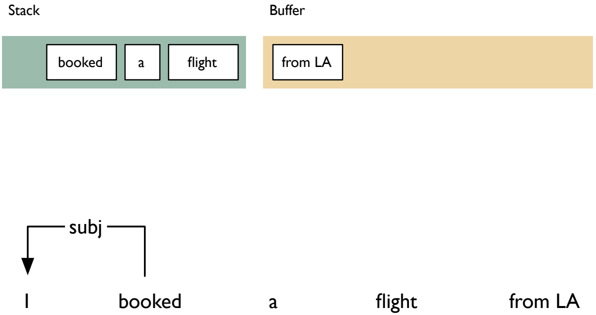
Example transition-based parsing : the arc-standard algorithm



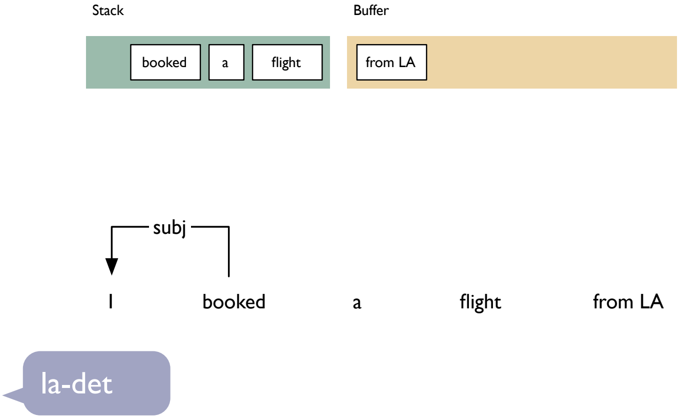
Example transition-based parsing : the arc-standard algorithm



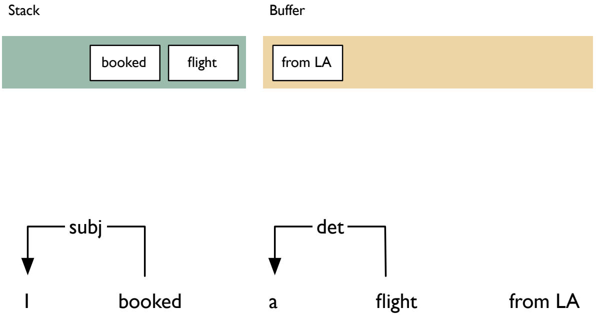
Example transition-based parsing : the arc-standard algorithm



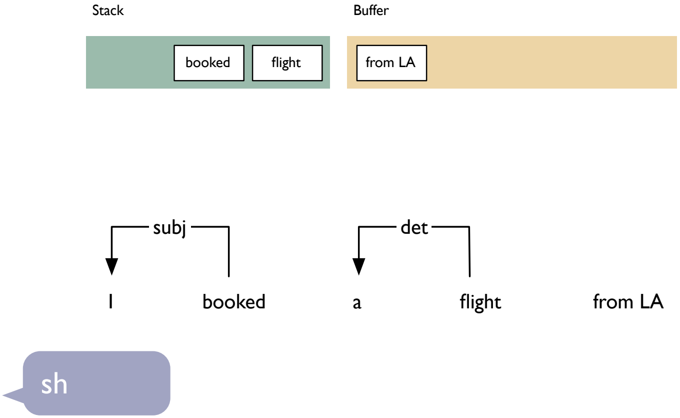
Example transition-based parsing : the arc-standard algorithm



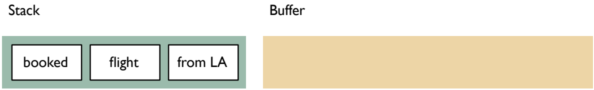
Example transition-based parsing : the arc-standard algorithm

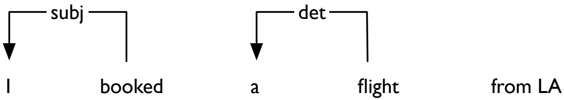


Example transition-based parsing : the arc-standard algorithm

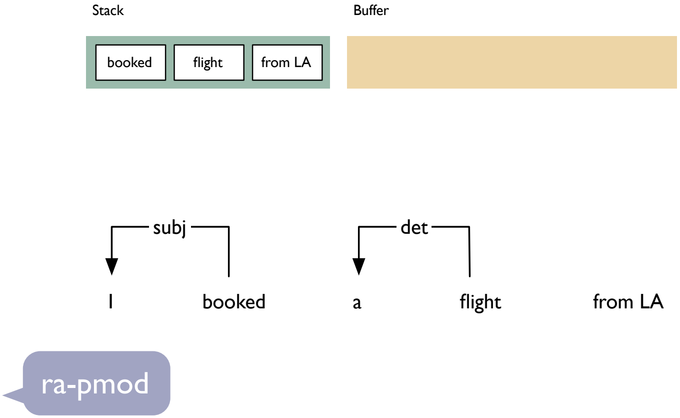


Example transition-based parsing : the arc-standard algorithm

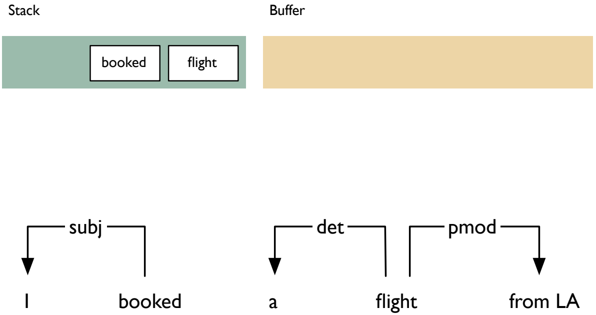




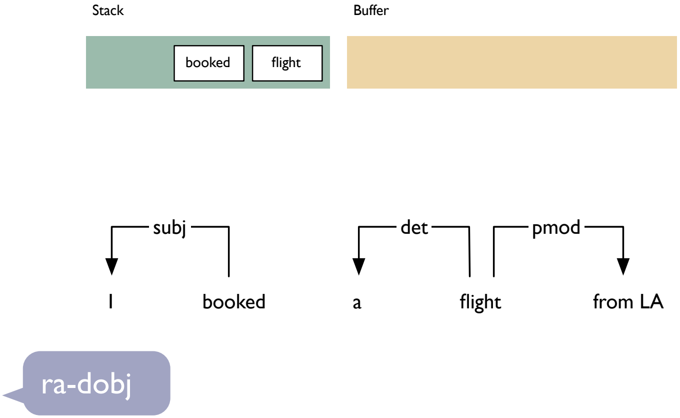
Example transition-based parsing : the arc-standard algorithm



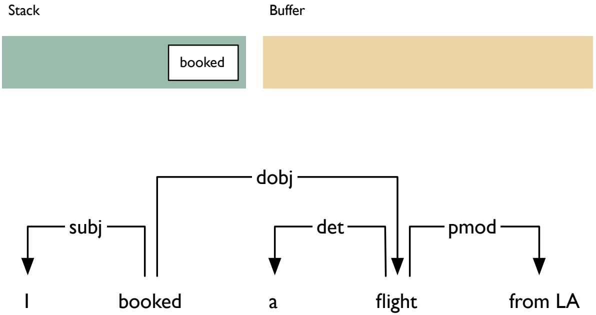
Example transition-based parsing : the arc-standard algorithm



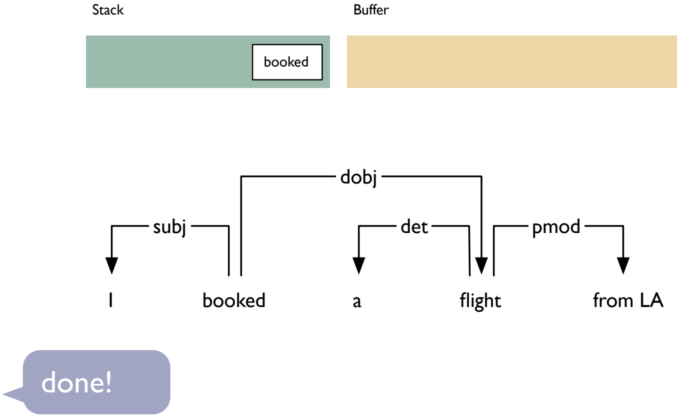
Example transition-based parsing : the arc-standard algorithm



Example transition-based parsing : the arc-standard algorithm



Example transition-based parsing :



How do we select next action ?

Train a classifier to predict next action from current configuration.