Natural Language Processing
Lecture 07 Syntax Parsing; PCFG Parsing; Dependency Parsing; Parsing with NNs

Qun Liu, Valentin Malykh
Huawei Noah's Ark Lab

Spring 2020
A course delivered at MIPT, Moscow

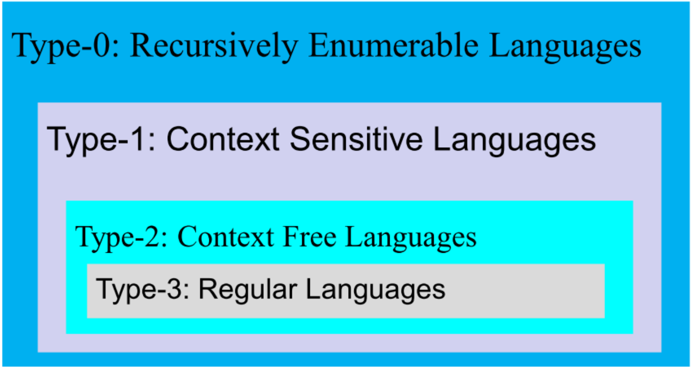
Content

Content

Chomsky hierarchy of grammars (Recap)

Grammar	Languages	Production Rules	Examples
Type 0	Recursively Enumerable	$\alpha A\beta \to \delta$	
Type 1	Context Sensitive	$\alpha A\beta \to \alpha\gamma\beta$	$L = \{\ a^n b^n c^n \mid n > 0\}$
Type 2	Context Free	$A \to \alpha$	$L = \{\ a^n b^n \mid n > 0\}$
Type 3	Regular	$A \to a$ or $A \to aB$	$L = \{\ a^n \mid n > 0\}$

Chomsky hierarchy (Recap)



Set inclusions described by the Chomsky hierarchy

Desirable Properties of a Grammar

Chomsky specified two properties that make a grammar "interesting and satisfying":

- It should be a finite specification of the strings of the language, rather than a list of its sentences.
- It should be revealing, in allowing strings to be associated with meaning (semantics) in a systematic way.

We can add another desirable property:

- It should capture structural and distributional properties of the language. (E.g. where heads of phrases are located; how a sentence transforms into a question; which phrases can float around the sentence.)

Desirable Properties of a Grammar

- Context-free grammars (CFGs) provide a pretty good approximation.
- Some features of NLs are more easily captured using mildly context-sensitive grammars, as well see later in the course.
- There are also more modern grammar formalisms that better capture structural and distributional properties of human languages. (E.g. combinatory categorial grammar.)
- Programming language grammars (such as the ones used with compilers, like LL(1)) aren't enough for NLs.

A Tiny Fragment of English

Let's say we want to capture in a grammar the structural and distributional properties that give rise to sentences like:

A duck walked in the park.	NP,V,PP
The man walked with a duck.	NP,V,PP
You made a duck.	Pro,V,NP
You made her duck.	? Pro,V,NP
A man with a telescope saw you.	NP,PP,V,Pro
A man saw you with a telescope.	NP,V,Pro,PP
You saw a man with a telescope.	Pro,V,NP,PP

We want to write grammatical rules that generate these phrase structures, and lexical rules that generate the words appearing in them.

Grammar for the Tiny Fragment of English

Grammar G1 generates the sentences on the previous slide:

Grammatical rules	**Lexical rules**
S $\rightarrow$ NP VP	Det $\rightarrow$ a $\mid$ the $\mid$ her (determiners)
NP $\rightarrow$ Det N	N $\rightarrow$ man $\mid$ park $\mid$ duck $\mid$ telescope (nouns)
NP $\rightarrow$ Det N PP	Pro $\rightarrow$ you (pronoun)
NP $\rightarrow$ Pro	V $\rightarrow$ saw $\mid$ walked $\mid$ made (verbs)
VP $\rightarrow$ V NP PP	Prep $\rightarrow$ in $\mid$ with $\mid$ for (prepositions)
VP $\rightarrow$ V NP	
VP $\rightarrow$ V	
PP $\rightarrow$ Prep NP	

Shay Cohen, Accelerated Natural Language Processing (Slides)

Context-free grammars: formal definition

A context-free grammar (CFG) $\mathcal{G}$ consists of

- a finite set N of non-terminals,
- a finite set Σ of terminals, disjoint from N,
- a finite set P of productions of the form $X \rightarrow \alpha$, where $X \in N$, $\alpha \in (N \cup \Sigma)^*$,
- a choice of start symbol $S \in N$.

A sentential form is any sequence of terminals and nonterminals that can appear in a derivation starting from the start symbol.

Formal definition: The set of sentential forms derivable from $\mathcal{G}$ is the smallest set $\mathcal{S}(\mathcal{G}) \subseteq (N \cup \Sigma)^*$ such that

- $S \in \mathcal{S}(\mathcal{G})$
- if $\alpha X \beta \in \mathcal{S}(\mathcal{G})$ and $X \rightarrow \gamma \ \in P$, then $\alpha\gamma\beta \in \mathcal{S}(\mathcal{G})$.

The language associated with grammar is the set of sentential forms that contain only terminals.

Formal definition: The language associated with $\mathcal{G}$ is defined by $\mathcal{L}(\mathcal{G}) \ = \ \mathcal{S}(\mathcal{G}) \cap \Sigma^*$

A sentential form is any sequence of terminals and nonterminals that can appear in a derivation starting from the start symbol.

Formal definition: The set of sentential forms derivable from $\mathcal{G}$ is the smallest set $\mathcal{S}(\mathcal{G}) \subseteq (N \cup \Sigma)^*$ such that

- $S \in \mathcal{S}(\mathcal{G})$
- if $\alpha X \beta \in \mathcal{S}(\mathcal{G})$ and $X \rightarrow \gamma \ \in P$, then $\alpha\gamma\beta \in \mathcal{S}(\mathcal{G})$.

The language associated with grammar is the set of sentential forms that contain only terminals.

Formal definition: The language associated with $\mathcal{G}$ is defined by $\mathcal{L}(\mathcal{G}) \ = \ \mathcal{S}(\mathcal{G}) \cap \Sigma^*$

A language $L \subseteq \Sigma^*$ is defined to be context-free if there exists some CFG $\mathcal{G}$ such that $L = \mathcal{L}(\mathcal{G})$.

Assorted remarks

- $X \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ is simply an abbreviation for a bunch of productions $X \to \alpha_1$, $X \to \alpha_2$, ..., $X \to \alpha_n$.
- These grammars are called context-free because a rule $X \to \alpha$ says that an X can *always* be expanded to α, no matter where the X occurs.
  This contrasts with context-sensitive rules, which might allow us to expand X only in certain contexts, e.g. $bXc \to b\alpha c$.
- Broad intuition: context-free languages allow nesting of structures to arbitrary depth. E.g. brackets, begin-end blocks, if-then-else statements, subordinate clauses in English, ...

Grammar for the Tiny Fragment of English

Grammar G1 generates the sentences on the previous slide:

Grammatical rules	**Lexical rules**
S → NP VP	Det → a | the | her (determiners)
NP → Det N	N → man | park | duck | telescope (nouns)
NP → Det N PP	Pro → you (pronoun)
NP → Pro	V → saw | walked | made (verbs)
VP → V NP PP	Prep → in | with | for (prepositions)
VP → V NP	
VP → V	
PP → Prep NP	

Does G1 produce a finite or an infinite number of sentences?

Recursion

Recursion in a grammar makes it possible to generate an infinite number of sentences.

In direct recursion, a non-terminal on the LHS of a rule also appears on its RHS. The following rules add direct recursion to G1:

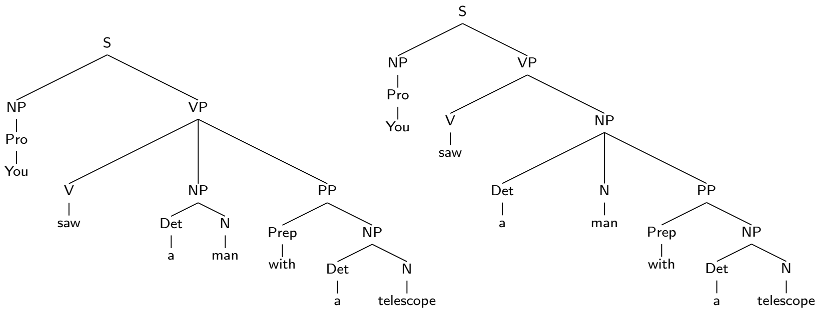
VP $\rightarrow$ VP Conj VP
Conj $\rightarrow$ and | or

In indirect recursion, some non-terminal can be expanded (via several steps) to a sequence of symbols containing that non-terminal:

NP $\rightarrow$ Det N PP
PP $\rightarrow$ Prep NP

Structural Ambiguity

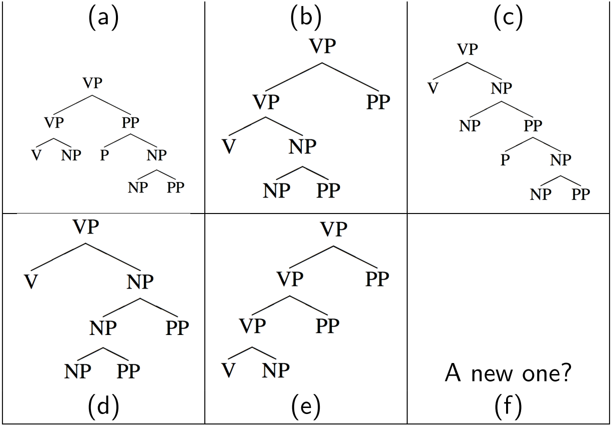
You saw a man with a telescope.



This illustrates attachment ambiguity: the PP can be a part of the VP or of the NP. Note that there's no POS ambiguity here.

Shay Cohen, Accelerated Natural Language Processing (Slides) 24 / 57

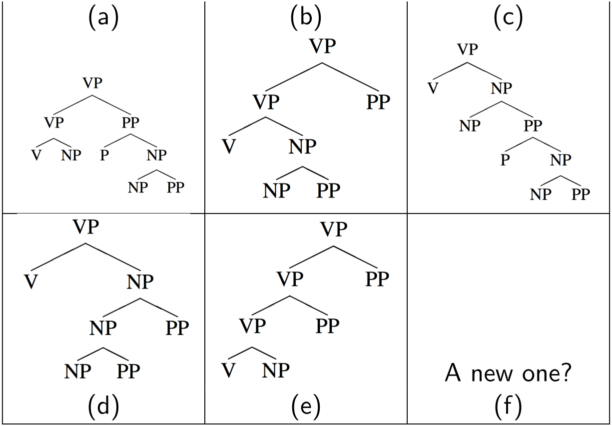
A Fun Exercise - Which is the VP?



saw the car from my house window with my telescope

Shay Cohen, Accelerated Natural Language Processing (Slides)

A Fun Exercise - Which is the VP?



saw the car from my house window with my telescope

E

Content

Chomsky Normal Form

A context-free grammar is in Chomsky normal form if all productions are of the form $A \rightarrow B\ C$ or $A \rightarrow a$ where A, B, C are nonterminals in the grammar and a is a word in the grammar.

Disregarding the empty string, every CFG is equivalent to a grammar in Chomsky normal form (the grammars' string languages are identical)

Why is that important?

- A normal form constrains the possible ways to represent an object
- Makes parsing efficient

Conversion to Chomsky Normal Form

- Replace all words in an RHS with a preterminal that rewrites to that word
- Break all RHSes into a sequence of RHSes with two nonterminals, possibly introducing new nonterminals:

$$S \rightarrow A_1 A_2 A_3$$

transforms into

$$S \rightarrow A_1 B$$

$$B \rightarrow A_2 A_3$$

.

Parsing algorithms

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

- 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, like 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.

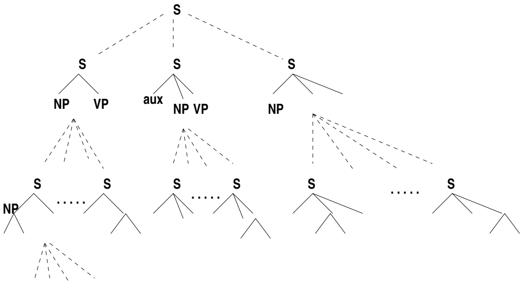
Parser properties

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



Shay Cohen, Accelerated Natural Language Processing (Slides)

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.)

Shay Cohen, Accelerated Natural Language Processing (Slides)

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:

S → NP VP	VP → V	NN → bit	V → bit
NP → DT NN	DT → the	NN → dog	V → 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 $\mathtt{NP} \rightarrow \mathtt{NP}\ \mathtt{PP}$, we get an infinite loop!

Shift-Reduce Parsing

A Shift-Reduce parser tries to find sequences of words and phrases that correspond to the righthand side of a grammar production and replace them with the lefthand side:

- **Directionality** = bottom-up: starts with the words of the input and tries to build trees from the words up.
- **Search strategy** = breadth-first: starts with the words, then applies rules with matching right hand sides, and so on until the whole sentence is reduced to an S.

Algorithm Sketch: Shift-Reduce Parsing

Until the words in the sentences are substituted with S:

- Scan through the input until we recognise something that corresponds to the RHS of one of the production rules (shift)
- Apply a production rule in reverse; i.e., replace the RHS of the rule which appears in the sentential form with the LHS of the rule (reduce)

A shift-reduce parser implemented using a stack:

1. start with an empty stack
2. a shift action pushes the current input symbol onto the stack
3. a reduce action replaces n items with a single item

Shay Cohen, Accelerated Natural Language Processing (Slides)

Shift-Reduce Parsing

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

Shay Cohen, Accelerated Natural Language Processing (Slides)

Shift-Reduce Parsing

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

Shay Cohen, Accelerated Natural Language Processing (Slides)

40 / 1

Shift-Reduce Parsing

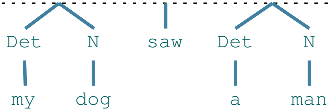
Stack	Remaining
NP Det N my dog	saw a man in the park

Shay Cohen, Accelerated Natural Language Processing (Slides)

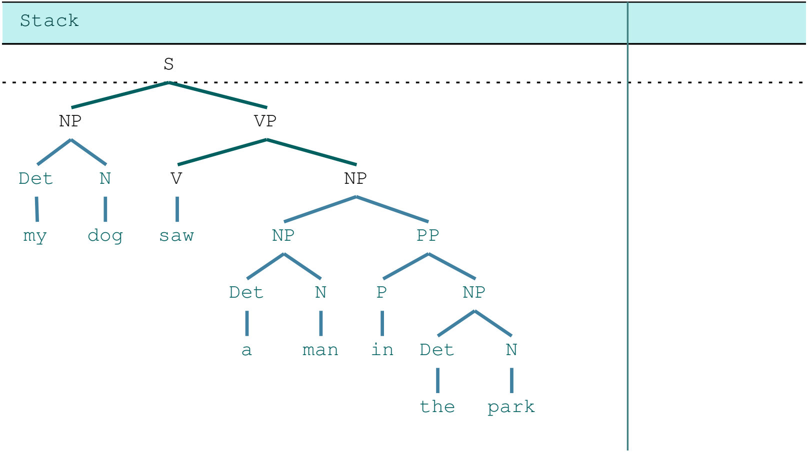
41 / 1

Shift-Reduce Parsing

Stack	Remaining
NP V NP	in the park

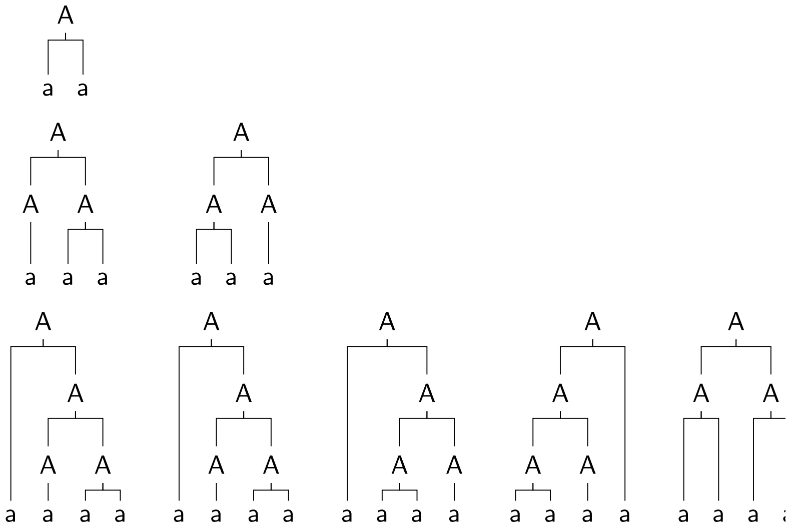


Shift-Reduce Parsing



Shay Cohen, Accelerated Natural Language Processing (Slides)

How many parses are there?



Shay Cohen, Accelerated Natural Language Processing (Slides)

How many parses are there?

Intution. Let $C(n)$ be the number of binary trees over a sentence of length n. The root of this tree has two subtrees: one over k words ($1 \leq k < n$), and one over $n - k$ words. Hence, for all values of k, we can combine any subtree over k words with any subtree over $n - k$ words:

$$C(n) = \sum_{k=1}^{n-1} C(k) \times C(n-k)$$

$$C(n) = \frac{(2n)!}{(n+1)!n!}$$

These numbers are called the Catalan numbers. They're big numbers!

n	1	2	3	4	5	6	8	9	10	11	12
$C(n)$	1	1	2	5	14	42	132	429	1430	4862	16796

Problems with Parsing as Search

1. A **recursive descent parser** (top-down) will do badly if there are many different rules for the same LHS. Hopeless for rewriting parts of speech (preterminals) with words (terminals).
2. A **shift-reduce parser** (bottom-up) does a lot of useless work: many phrase structures will be locally possible, but globally impossible. Also inefficient when there is much lexical ambiguity.
3. Both strategies do repeated work by re-analyzing the same substring many times.

We will see how chart parsing solves the re-parsing problem, and also copes well with ambiguity.

Dynamic Programming

With a CFG, a parser should be able to avoid re-analyzing sub-strings because the analysis of any sub-string is independent of the rest of the parse.

$$\underbrace{\text{The dog}}_{\text{NP}}\ \text{saw}\ \overbrace{\underbrace{\text{a man}}_{\text{NP}}\ \text{in}\ \underbrace{\text{the park}}_{\text{NP}}}^{\text{NP}}$$

The parser's exploration of its search space can exploit this independence if the parser uses dynamic programming.

Dynamic programming is the basis for all chart parsing algorithms.

Shay Cohen, Accelerated Natural Language Processing (Slides)

Parsing as Dynamic Programming

- Given a problem, systematically fill a table of solutions to sub-problems: this is called memoization.
- Once solutions to all sub-problems have been accumulated, solve the overall problem by composing them.
- For parsing, the sub-problems are analyses of sub-strings and correspond to constituents that have been found.
- Sub-trees are stored in a chart (aka well-formed substring table), which is a record of all the substructures that have ever been built during the parse.

Solves **re-parsing problem**: sub-trees are looked up, not re-parsed!
Solves **ambiguity problem**: chart implicitly stores all parses!

Depicting a Chart

A chart can be depicted as a matrix:

- Rows and columns of the matrix correspond to the start and end positions of a span (ie, starting right before the first word, ending right after the final one);
- A cell in the matrix corresponds to the sub-string that starts at the row index and ends at the column index.
- It can contain information about the type of constituent (or constituents) that span(s) the substring, pointers to its sub-constituents, and/or predictions about what constituents might follow the substring.

CYK Algorithm

CYK (Cocke, Younger, Kasami) is an algorithm for recognizing and recording constituents in the chart.

- Assumes that the grammar is in Chomsky Normal Form: rules all have form $A \rightarrow BC$ or $A \rightarrow w$.
- Conversion to CNF can be done automatically.

NP	$\rightarrow$	Det Nom	NP	$\rightarrow$	Det Nom
Nom	$\rightarrow$	N | OptAP Nom	Nom	$\rightarrow$	*book* | *orange* | AP Nom
OptAP	$\rightarrow$	ϵ | OptAdv A	AP	$\rightarrow$	*heavy* | *orange* | Adv A
A	$\rightarrow$	*heavy* | *orange*	A	$\rightarrow$	*heavy* | *orange*
Det	$\rightarrow$	*a*	Det	$\rightarrow$	*a*
OptAdv	$\rightarrow$	ϵ | *very*	Adv	$\rightarrow$	*very*
N	$\rightarrow$	*book* | *orange*			

Shay Cohen, Accelerated Natural Language Processing (Slides)

CYK: an example

Let's look at a simple example before we explain the general case.

Grammar Rules in CNF

NP	$\rightarrow$	Det Nom
Nom	$\rightarrow$	*book* | *orange* | AP Nom
AP	$\rightarrow$	*heavy* | *orange* | Adv A
A	$\rightarrow$	*heavy* | *orange*
Det	$\rightarrow$	*a*
Adv	$\rightarrow$	*very*

(N.B. Converting to CNF sometimes breeds duplication!)

Now let's parse: *a very heavy orange book*

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

	1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0 a					
1 very					
2 heavy					
3 orange					
4 book					

Shay Cohen, Accelerated Natural Language Processing (Slides)

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

	1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0 a	Det				
1 very					
2 heavy					
3 orange					
4 book					

Shay Cohen, Accelerated Natural Language Processing (Slides)

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

	1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0 a	Det				
1 very		Adv			
2 heavy					
3 orange					
4 book					

Shay Cohen, Accelerated Natural Language Processing (Slides)

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

	1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0 a	Det				
1 very		Adv			
2 heavy			A,AP		
3 orange					
4 book					

Shay Cohen, Accelerated Natural Language Processing (Slides)