Syntactic Parsing

Constituency, Dependency, Top-down and Bottom-up Parsing

Complete study notes: definitions, key points, architecture, step-by-step examples and applications

Contents

1. Introduction to Syntactic Parsing
2. Constituency Parsing
3. Dependency Parsing
4. Top-down Parsing
5. Bottom-up Parsing
6. Comparison and Quick Revision

1. Introduction to Syntactic Parsing

Definition

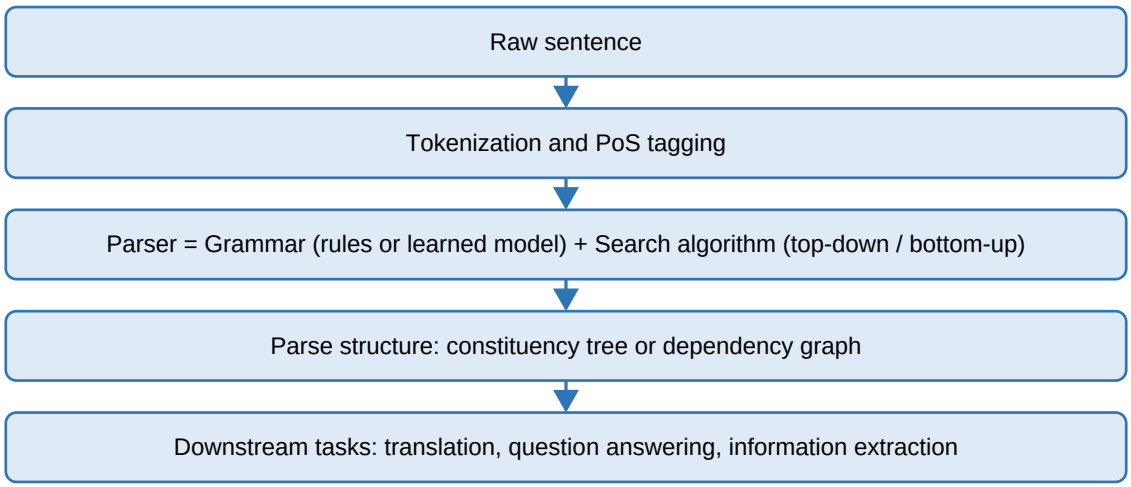
Syntactic parsing (syntax analysis) is the process of analysing a sentence to find its grammatical structure according to a grammar. The result is a **parse tree** or **parse graph** that shows how words group into phrases or how words depend on one another.

> **Input:** The dog chased the cat
> **Output (bracketed parse):** (S (NP (Det The) (N dog)) (VP (V chased) (NP (Det the) (N cat))))

Important points

- A parser needs two things: a **grammar** (rules or a learned model) and a **search algorithm** that applies the grammar to the sentence.
- **Two ways to represent structure:** constituency (phrase structure) and dependency (word-to-word relations).
- **Two search strategies:** top-down (start from the goal symbol S) and bottom-up (start from the words).
- **Ambiguity** is the main difficulty: a sentence may have several valid parses, and the parser must find all of them or choose the best one.
- Parsing usually follows PoS tagging, which supplies the word categories.
- Parsers are evaluated by comparing their output with a **treebank** (e.g., Penn Treebank, Universal Dependencies).

Architecture: general parsing pipeline



Two independent choices

Choice	Options	Question it answers
Representation (what is built)	Constituency tree, dependency graph	What does the output look like?
Strategy (how it is built)	Top-down, bottom-up (also mixed, e.g., Earley, left-corner)	In what order are the structures searched?

Applications of parsing

- Machine translation, grammar checking and text simplification.
- Information extraction, relation extraction and question answering.
- Semantic analysis (semantic role labelling), sentiment analysis and dialogue systems.
- Compilers and interpreters for programming languages (the same techniques).

2. Constituency Parsing

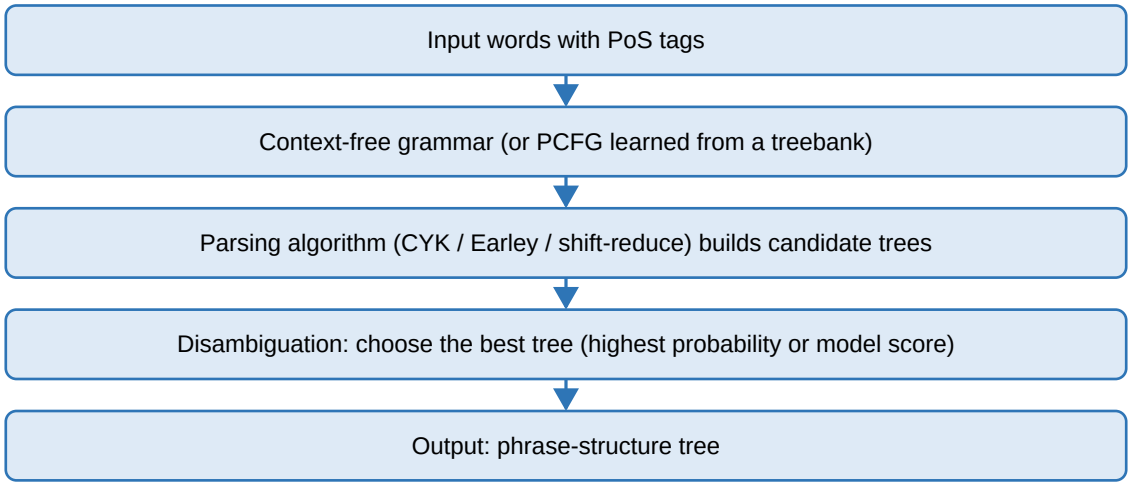
Definition

Constituency parsing (phrase-structure parsing) breaks a sentence into nested **constituents**, groups of words that behave as a single unit, such as noun phrases (NP), verb phrases (VP) and prepositional phrases (PP). It uses a **context-free grammar (CFG)** G = (N, T, R, S): non-terminals N, terminals (words) T, rules R of the form A -> B C, and a start symbol S.

Important points

- A **constituent** can be tested: it can be replaced by a pronoun, moved as a block, or used as an answer to a question (“The dog” can be replaced by “it”).
- The tree has **non-terminal nodes** (S, NP, VP) and **pre-terminal nodes** (the PoS tags) above the words (leaves).
- **Grammar rules** are written as S -> NP VP, NP -> Det N, VP -> V NP, and so on. Lexical rules link PoS tags to words (Det -> the).
- **PCFG (probabilistic CFG)** attaches a probability to each rule. The probability of a tree is the product of its rule probabilities, and the parser selects the tree with the highest probability.
- **Ambiguity:** the same sentence may have several trees (PP attachment is the classic case).
- **Algorithms:** CYK (bottom-up dynamic programming), Earley (mixed), shift-reduce, and neural span-based parsers.
- **Evaluation:** PARSEVAL labelled precision, recall and F1 over constituent brackets.

Architecture



Example 1: building the constituency tree (step by step)

Sentence: **“The dog chased the cat.”** Grammar:

Rule type	Rules
Phrase rules	S -> NP VP; NP -> Det N; VP -> V NP
Lexical rules	Det -> the; N -> dog | cat; V -> chased

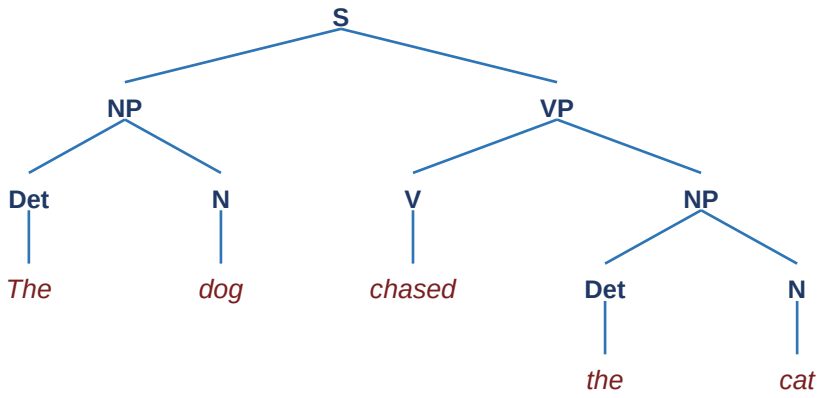
Step 1: PoS tags are assigned: The/Det dog/N chased/V the/Det cat/N.

Step 2: Group Det + N into noun phrases: NP(The dog) and NP(the cat), using NP -> Det N.

Step 3: Group V + NP into a verb phrase: VP(chased the cat), using VP -> V NP.

Step 4: Group NP + VP into a sentence: S(The dog chased the cat), using S -> NP VP.

Step 5: Output the tree (below) or the bracketed form (S (NP (Det The) (N dog)) (VP (V chased) (NP (Det the) (N cat)))).



Example 2: structural ambiguity (PP attachment)

Sentence: **“I saw the man with a telescope.”** Both trees are valid under the same grammar (VP -> VP PP and NP -> NP PP).

Parse	Bracketing	Meaning
A: PP attaches to VP	(VP (VP saw the man) (PP with a telescope))	I used a telescope to see the man.
B: PP attaches to NP	(VP saw (NP (NP the man) (PP with a telescope)))	I saw the man who had a telescope.

A PCFG or a neural model scores both trees and returns the more probable one. World knowledge or context is needed to be sure.

Applications

- Grammar checking and syntax-based machine translation.
- Extracting noun phrases and key phrases for information extraction and indexing.
- Semantic role labelling, question answering and phrase-level sentiment analysis.

3. Dependency Parsing

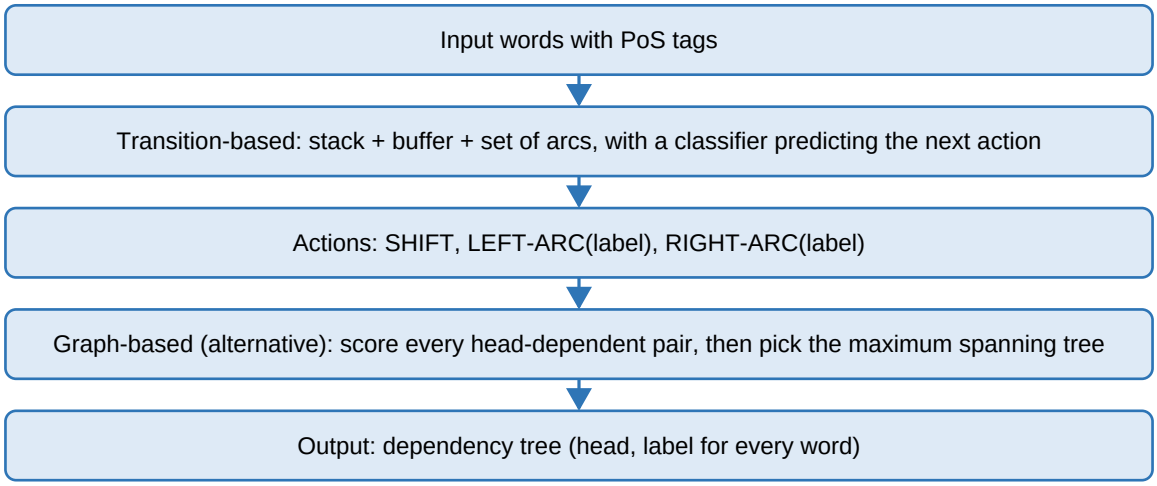
Definition

Dependency parsing analyses the grammatical structure of a sentence as a set of **directed, labelled relations between words**. Every word (the *dependent*) is linked to exactly one other word (its *head*), except the main verb, which is attached to an artificial **ROOT**. There are no phrase nodes.

Important points

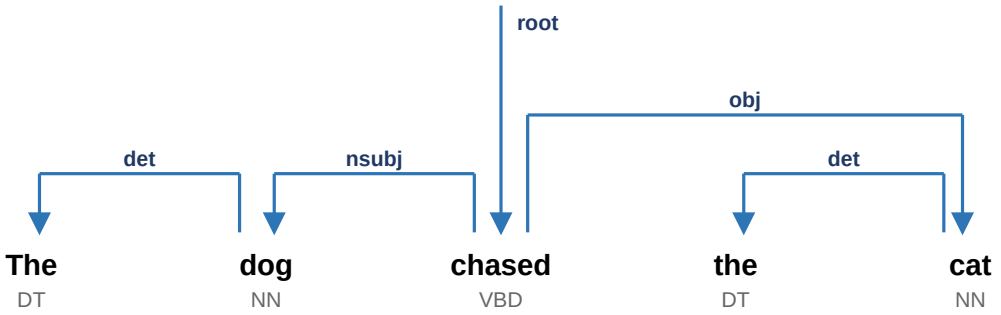
- **One head per word:** the result is a tree with n words, n arcs and a single root.
- **Typed relations** (Universal Dependencies): nsubj (subject), obj (object), det (determiner), amod (adjective modifier), nmod (noun modifier), advmod (adverb modifier), root.
- **The verb is the centre** of the clause; its arguments depend on it directly.
- **Projective vs non-projective:** a tree is projective if arcs do not cross. Languages with free word order (Telugu, Hindi, Czech) need non-projective parsing.
- **Algorithms:** *transition-based* (arc-standard, arc-eager: a stack, a buffer and a classifier choose actions; fast, linear time) and *graph-based* (score all possible arcs and find the maximum spanning tree with Chu-Liu-Edmonds).
- **Evaluation:** UAS (unlabelled attachment score, correct heads) and LAS (labelled attachment score, correct heads and labels).

Architecture



Example 1: dependency tree of a sentence

Sentence: **“The dog chased the cat.”**



Index	Word	Head	Relation	Meaning
1	The	dog (2)	det	determiner of “dog”
2	dog	chased (3)	nsubj	subject of the verb
3	chased	ROOT (0)	root	main verb of the sentence
4	the	cat (5)	det	determiner of “cat”
5	cat	chased (3)	obj	object of the verb

Example 2: transition-based parsing, arc-standard (step by step)

The parser keeps a **stack** (starting with ROOT) and a **buffer** (the input words). **SHIFT** moves the next word to the stack. **LEFT-ARC** makes the second-from-top word depend on the top word and removes it. **RIGHT-ARC** makes the top word depend on the second-from-top word and removes it.

Step	Action	Stack	Buffer	Arc added
1	SHIFT	[ROOT, The]	[dog, chased, the, cat]	-
2	SHIFT	[ROOT, The, dog]	[chased, the, cat]	-
3	LEFT-ARC (det)	[ROOT, dog]	[chased, the, cat]	dog -> The
4	SHIFT	[ROOT, dog, chased]	[the, cat]	-
5	LEFT-ARC (nsubj)	[ROOT, chased]	[the, cat]	chased -> dog
6	SHIFT	[ROOT, chased, the]	[cat]	-
7	SHIFT	[ROOT, chased, the, cat]	[ ]	-
8	LEFT-ARC (det)	[ROOT, chased, cat]	[ ]	cat -> the
9	RIGHT-ARC (obj)	[ROOT, chased]	[ ]	chased -> cat
10	RIGHT-ARC (root)	[ROOT]	[ ]	ROOT -> chased

The buffer is empty and only ROOT is on the stack, so parsing is complete. The 10 steps produced exactly the 5 arcs of the dependency tree, in time linear in the sentence length.

Applications

- Relation extraction and open information extraction (subject-verb-object triples).
- Machine translation and question answering that rely on predicate-argument structure.
- Parsing free-word-order and low-resource languages (Telugu, Hindi), using Universal Dependencies treebanks.
- Text simplification, semantic role labelling and sentiment targets (which noun an adjective modifies).

4. Top-down Parsing

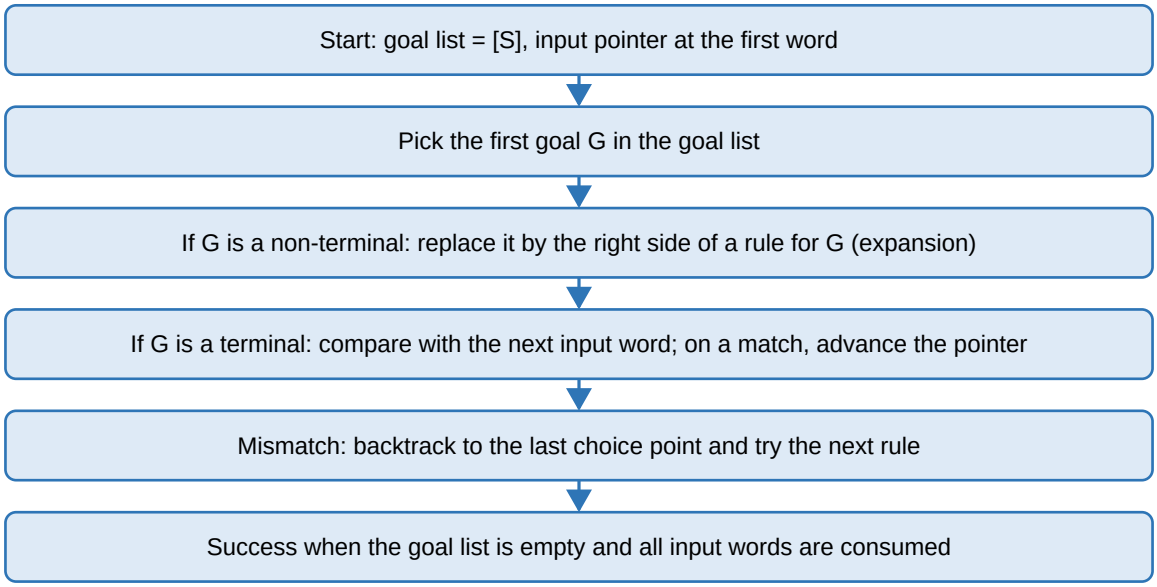
Definition

Top-down parsing is a **goal-driven** strategy. It starts with the start symbol S and repeatedly **expands** non-terminals using the grammar rules, trying to build a tree that matches the input words from left to right. It is also called **recursive-descent** or predictive parsing.

Important points

- **Direction:** root to leaves. The parser predicts what it expects to see, then checks the input.
- **Never builds a constituent that cannot be part of an S**, because every expansion starts from the goal.
- **Backtracking:** when a chosen rule fails to match the input, the parser returns to the last choice point and tries the next rule.
- **Wasted work:** it may predict many structures that the input words do not support (e.g., trying NP -> Pronoun when the word is “the”), and it may repeat the same sub-parse after backtracking.
- **Left recursion problem:** a rule such as NP -> NP PP makes the parser expand NP forever. Solutions: rewrite the grammar to remove left recursion, limit depth, or use chart-based methods.
- **Improvements:** look-ahead (LL(1) predictive parsing), memoization (packrat), and the Earley algorithm, which combines top-down prediction with chart storage.

Architecture



Example: top-down parse with backtracking (step by step)

Sentence: **“the dog chased the cat”**. Grammar:

Rules
S -> NP VP; NP -> Pron | Det N; VP -> V NP | V
Pron -> he; Det -> the; N -> dog | cat; V -> chased

Step	Goal list (stack)	Action	Input read
1	[S]	Start with the goal S	0
2	[NP, VP]	Expand S -> NP VP	0
3	[Pron, VP]	Expand NP -> Pron (first rule)	0
4	[he, VP]	Expand Pron -> he; “he” does not match “the”: **FAIL, backtrack**	0
5	[Det, N, VP]	Try NP -> Det N (second rule)	0
6	[N, VP]	Det -> the matches “the”	1
7	[VP]	N -> dog matches “dog”	2
8	[V, NP]	Expand VP -> V NP	2
9	[NP]	V -> chased matches “chased”	3
10	[Pron]	Try NP -> Pron: “he” does not match “the”: **FAIL, backtrack**	3
11	[Det, N]	Try NP -> Det N; Det -> the matches “the”	4
12	[N]	N -> dog: “dog” does not match “cat”: **FAIL, backtrack**	4
13	[ ]	N -> cat matches “cat”. Goal list empty and all 5 words read: **SUCCESS**	5

Result: the tree is the same as the one in Section 2. The parser made three wrong guesses (steps 3, 10 and 12) and had to backtrack, which shows the main cost of the top-down strategy.

Applications

- Recursive-descent parsers for programming languages and configuration files.
- Grammar-driven systems with small controlled grammars, such as voice commands and dialogue systems.
- Prediction in speech recognition and language generation (what can come next); Prolog DCG grammars.
- Earley and left-corner parsers for natural language, which add top-down prediction to chart parsing.

5. Bottom-up Parsing

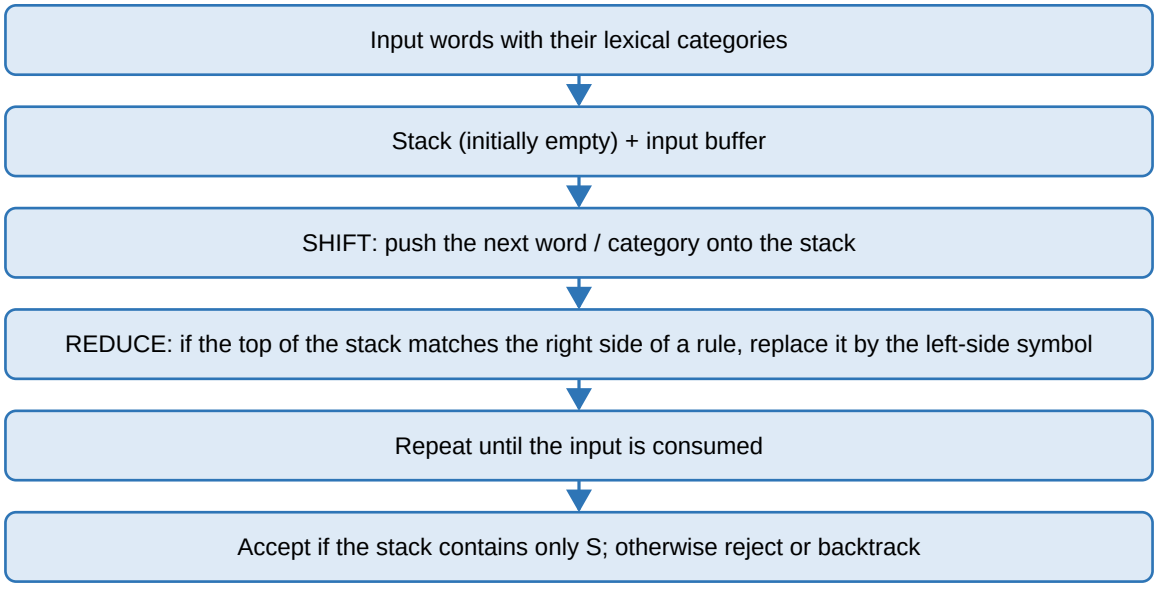
Definition

Bottom-up parsing is a **data-driven** strategy. It starts from the input words, assigns them their categories, and repeatedly **reduces** sequences of symbols to a non-terminal (using the grammar rules in reverse) until a single S covering the whole sentence is reached. Two main methods are **shift-reduce** parsing and the **CYK algorithm**.

Important points

- **Direction:** leaves to root. Every structure built is supported by the actual words.
- **Shift-reduce** uses a stack and an input buffer. SHIFT moves the next word onto the stack; REDUCE replaces the top symbols matching the right side of a rule by its left side.
- **Conflicts:** shift-shift, shift-reduce and reduce-reduce conflicts occur when more than one action is possible; they are resolved by look-ahead, backtracking or a trained classifier.
- **Wasted work:** it may build constituents that can never become part of a complete S (e.g., an NP that is later unusable).
- **CYK algorithm:** a dynamic-programming chart parser for grammars in Chomsky normal form (rules A -> B C or A -> word). It fills a triangular table for every span, with time $O(n^3 |G|)$.
- **Probabilistic CYK** stores the best-scoring subtree for each cell and gives the most probable PCFG parse.

Architecture



Example 1: shift-reduce parse (step by step)

Sentence: **“the dog chased the cat”**, with the grammar of Section 2.

Step	Stack	Remaining input	Action
1	[ ]	the dog chased the cat	SHIFT the
2	[the]	dog chased the cat	REDUCE Det -> the

Step	Stack	Remaining input	Action
3	[Det]	dog chased the cat	SHIFT dog
4	[Det, dog]	chased the cat	REDUCE N -> dog
5	[Det, N]	chased the cat	REDUCE NP -> Det N
6	[NP]	chased the cat	SHIFT chased
7	[NP, chased]	the cat	REDUCE V -> chased
8	[NP, V]	the cat	SHIFT the, then REDUCE Det -> the
9	[NP, V, Det]	cat	SHIFT cat
10	[NP, V, Det, cat]	-	REDUCE N -> cat
11	[NP, V, Det, N]	-	REDUCE NP -> Det N
12	[NP, V, NP]	-	REDUCE VP -> V NP
13	[NP, VP]	-	REDUCE S -> NP VP
14	[S]	-	**ACCEPT**: stack holds S and the input is empty

Example 2: CYK parsing (step by step)

The grammar of Section 2 is already in Chomsky normal form. Positions: 1 the, 2 dog, 3 chased, 4 the, 5 cat. Cell (i, k) lists the non-terminals that cover the words from i to k.

Step 1: Span length 1 (lexical rules): the = Det, dog = N, chased = V, the = Det, cat = N.

Step 2: Span length 2: (1-2) “the dog”: Det + N matches NP -> Det N, so NP. (4-5) “the cat”: Det + N, so NP. (2-3) N + V and (3-4) V + Det match no rule.

Step 3: Span length 3: (3-5) “chased the cat”: V (3) + NP (4-5) matches VP -> V NP, so VP. (1-3) and (2-4) have no rule.

Step 4: Span length 4: (1-4) and (2-5) have no valid split, so they stay empty.

Step 5: Span length 5: (1-5): split after word 2, NP (1-2) + VP (3-5) matches S -> NP VP, so **S**. S in the top cell means the sentence is grammatical.

Span length	Start 1 (the)	Start 2 (dog)	Start 3 (chased)	Start 4 (the)	Start 5 (cat)
1	Det	N	V	Det	N
2	NP	-	-	NP	
3	-	-	VP		
4	-	-			
5	**S**				

Applications

- Compilers and interpreters (yacc/bison use LR shift-reduce parsing).
- Statistical constituency parsers: probabilistic CYK with PCFGs, and modern shift-reduce and neural span parsers.

- Robust parsing of real text, since structures are built only from words actually present.
- Transition-based dependency parsers use the same shift/reduce idea (Section 3).

6. Comparison and Quick Revision

Constituency vs dependency parsing

Aspect	Constituency	Dependency
Structure	Nested phrases (NP, VP, S)	Word-to-word head-dependent arcs
Nodes	Words plus non-terminal nodes	Words only
Grammar	Context-free grammar / PCFG	Dependency grammar; relation labels
Strengths	Shows phrase boundaries and hierarchy	Compact, shows who does what to whom; good for free word order
Typical algorithms	CYK, Earley, shift-reduce	Transition-based, graph-based (MST)
Evaluation	PARSEVAL F1 on brackets	UAS / LAS
Resources	Penn Treebank	Universal Dependencies

Top-down vs bottom-up parsing

Aspect	Top-down	Bottom-up
Starting point	Start symbol S (the goal)	Input words
Driving force	Goal-driven (grammar predicts)	Data-driven (words build structure)
Operation	Expand non-terminals	Shift and reduce (or fill chart cells)
Strength	Never builds structures that cannot lead to S	Never predicts structures that the words do not support
Weakness	Wasted predictions; backtracking; left recursion loops	Builds useless constituents; shift-reduce conflicts
Examples	Recursive descent, LL, DCG	Shift-reduce, LR, CYK
Hybrid	Earley and left-corner parsers combine both	

Quick revision points

- **Syntactic parsing** finds the grammatical structure of a sentence using a grammar and a search algorithm.
- **Constituency:** CFG phrase tree (S, NP, VP); PCFG picks the most probable tree; CYK and Earley are classic algorithms.
- **Dependency:** every word has one head; labelled arcs; transition-based (stack, buffer, SHIFT / LEFT-ARC / RIGHT-ARC) or graph-based (MST).
- **Top-down:** start from S and expand rules; uses backtracking; fails on left recursion.
- **Bottom-up:** start from words and reduce to S; shift-reduce with a stack; CYK fills a table in $O(n^3)$.

Summary: Constituency and dependency parsing describe *what* structure a sentence has, while top-down and bottom-up parsing describe *how* the parser searches for it. A parser combines one representation with one strategy, for example a bottom-up CYK constituency parser or a transition-based dependency parser.