Parsing

1. Grammars and parsing
2. Top-down and bottom-up parsing
3. Chart parsers
4. Bottom-up chart parsing
5. The Earley Algorithm

Slide CS474–1

Syntax

syntax: from the Greek *syntaxis*, meaning "setting out together or arrangement."
Refers to the way words are arranged together.

Why worry about syntax?

- The boy ate the frog.
- The frog was eaten by the boy.
- The frog that the boy ate died.
- The boy whom the frog was eaten by died.

Slide CS474–2

Syntactic Analysis

Key ideas:

- **constituency**: groups of words may behave as a single unit or phrase
- **grammatical relations**: refer to the SUBJECT, OBJECT, INDIRECT OBJECT, etc.
- **subcategorization and dependencies**: refer to certain kinds of relations between words and phrases, e.g. *want* can be followed by an infinitive, but *find* and *work* cannot.

All can be modeled by various kinds of grammars that are based on context-free grammars.

Slide CS474–3

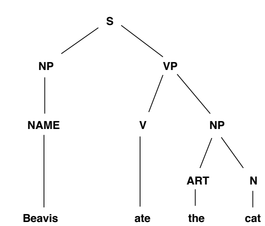
Grammars and Parsing

Need a **grammar:** a formal specification of the structures allowable in the language.

Need a **parser**: algorithm for assigning syntactic structure to an input sentence.

Sentence | **Parse Tree**

Beavis ate the cat.



Slide CS474–4

CFG example

CFG's are also called phrase-structure grammars.
Equivalent to Backus-Naur Form (BNF).

1. S → NP VP
2. VP → V NP
3. NP → NAME
4. NP → ART N
5. NAME → Beavis
6. V → ate
7. ART → the
8. N → cat

- CFG's are *powerful* enough to describe most of the structure in natural languages.
- CFG's are *restricted* enough so that efficient parsers can be built.

Slide CS474–5

CFG's

A context free grammar consists of:

1. a set of non-terminal symbols N
2. a set of terminal symbols Σ (disjoint from N)
3. a set of productions, P, each of the form $A \rightarrow \alpha$, where A is a non-terminal and α is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$
4. a designated start symbol S

Slide CS474–6

Derivations

- If the rule $A \rightarrow \beta \in P$, and α and γ are strings in the set $(\Sigma \cup N)^*$, then we say that $\alpha A \gamma$ **directly derives** $\alpha\beta\gamma$, or $\alpha A \gamma \Rightarrow \alpha\beta\gamma$
- Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be strings in $(\Sigma \cup N)^*$, $m > 1$, such that

$$\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m,$$

then we say that α_1 **derives** α_m or $\alpha_1 \overset{*}{\Rightarrow} \alpha_m$

Slide CS474–7

L_G

The language L_G generated by a grammar G is the set of strings composed of terminal symbols that can be derived from the designated start symbol S.

$$L_G = \{w | w \in \Sigma^*, S \overset{*}{\Rightarrow} w\}$$

Parsing: the problem of mapping from a string of words to its parse tree according to a grammar G.

Slide CS474–8

General Parsing Strategies

Grammar	Top-Down	Bottom-Up
1. S → NP VP	S → NP VP	→ NAME ate the cat
2. VP → V NP	→ NAME VP	→ NAME V the cat
3. NP → NAME	→ Beav VP	→ NAME V ART cat
4. NP → ART N	→ Beav V NP	→ NAME V ART N
5. NAME → Beavis	→ Beav ate NP	→ NP V ART N
6. V → ate	→ Beav ate ART N	→ NP V NP
7. ART → the	→ Beav ate the N	→ NP VP
8. N → cat	→ Beav ate the cat	→ S

A Top-Down Parser

Input: CFG grammar, lexicon, sentence to parse
Output: yes/no

State of the parse: (*symbol list*, *position*)

$_1$ The $_2$ old $_3$ man $_4$ cried $_5$

start state: ((S) 1)

Grammar and Lexicon

Grammar:

1. S → NP VP
2. NP → art n
3. NP → art adj n
4. VP → v
5. VP → v NP

Lexicon:
the: art
old: adj, n
man: n, v
cried: v

$_1$ The $_2$ old $_3$ man $_4$ cried $_5$

Algorithm for a Top-Down Parser

$PSL \leftarrow$ (((S) 1))

1. *Check for failure.* If PSL is empty, return NO.
2. *Select the current state, C.* C ← pop (PSL).
3. *Check for success.* If C = (() <final-position>), YES.
4. *Otherwise, generate the next possible states.*
   (a) $s_1 \leftarrow$ first-symbol(C)
   (b) If s_1 is a *lexical symbol* and next word can be in that class, create new state by removing s_1, updating the word position, and adding it to PSL. (I'll add to front.)
   (c) If s_1 is a *non-terminal*, generate a new state for each rule in the grammar that can rewrite s_1. Add all to PSL. (Add to front.)

Example

Current state	Backup states
1. ((S) 1)	
2. ((NP VP) 1)	
3. ((art n VP) 1)	((art adj n VP) 1)
4. ((n VP) 2)	((art adj n VP) 1)
5. ((VP) 3)	((art adj n VP) 1)
6. ((v) 3)	((v NP) 3) ((art adj n VP) 1)
7. (() 4)	((v NP) 3) ((art adj n VP) 1) Backtrack

Slide CS474–13

8. ((v NP) 3)	((art adj n VP) 1) leads to backtracking
...	
9. ((art adj n VP) 1)	
10. ((adj n VP) 2)	
11. ((n VP) 3)	
12. ((VP) 4)	
13. ((v) 4)	((v NP) 4)
14. (() 5)	((v NP) 4)
YES	

DONE!

Slide CS474–14

Problems with the Top-Down Parser

1. Only judges grammaticality.
2. Stops when it finds a single derivation.
3. No semantic knowledge employed.
4. No way to rank the derivations.
5. Problems with left-recursive rules.
6. Problems with ungrammatical sentences.

Slide CS474–15

Efficient Parsing

The top-down parser is terribly inefficient.

Have the first year Phd students in the computer science department take the Q-exam.

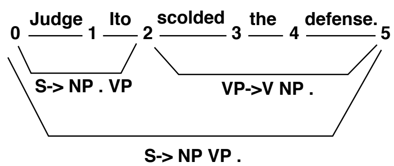
Have the first year Phd students in the computer science department taken the Q-exam?

Slide CS474–16

Chart Parsers

chart: data structure that stores partial results of the parsing process in such a way that they can be reused. The chart for an n-word sentence consists of:

- $n + 1$ **vertices**
- a number of **edges** that connect vertices



Slide CS474–17

Chart Parsing: The General Idea

The process of parsing an n-word sentence consists of forming a chart with $n + 1$ vertices and adding edges to the chart one at a time.

- Goal: To produce a complete edge that spans from vertex 0 to n and is of category S.
- There is no backtracking.
- Everything that is put in the chart stays there.
- Chart contains all information needed to create parse tree.

Slide CS474–18

Bottom-UP Chart Parsing Algorithm

Do until there is no input left:

1. If the agenda is empty, get next word from the input, look up word categories, add to agenda (as constituent spanning two postions).
2. Select a constituent from the agenda: constituent C from p_1 to p_2.
3. Insert C into the chart from position p_1 to p_2.
4. For each rule in the grammar of form $X \rightarrow C\ X_1 \ldots X_n$, add an active edge of form $X \rightarrow C \circ X_1 \ldots X_n$ from p_1 to p_2.

Slide CS474–19

5. Extend existing edges that are looking for a C.
   (a) For any active edge of form $X \rightarrow X_1 \ldots \circ C X_n$ from p_0 to p_1, add a new active edge $X \rightarrow X_1 \ldots C \circ X_n$ from p_0 to p_2.
   (b) For any active edge of form $X \rightarrow X_1 \ldots X_n \circ C$ from p_0 to p_1, add a new (completed) constituent of type X from p_0 to p_2 to the agenda.

Slide CS474–20

Grammar and Lexicon

Grammar:

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V NP

Lexicon:

the: ART

old: ADJ, N

man: N, V

boat: N

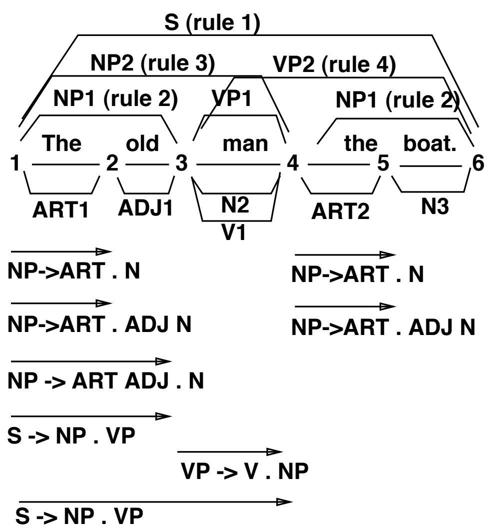
Sentence: $_1$ The $_2$ old $_3$ man $_4$ the $_5$ boat $_6$

Slide CS474–21

Example

[See .ppt slides]

Slide CS474–22



Slide CS474–23

Bottom-up Chart Parser

Is it any less naive than the top-down parser?

1. Only judges grammaticality.[fixed]
2. Stops when it finds a single derivation.[fixed]
3. No semantic knowledge employed.
4. No way to rank the derivations.
5. Problems with ungrammatical sentences.[better]
6. Terribly inefficient.

Slide CS474–24

Efficient Parsing

$n =$ sentence length
Time complexity for naive algorithm: exponential in n
Time complexity for bottom-up chart parser: $\bigcirc(n^3)$

Options for improving efficiency:

1. Don't do twice what you can do once.
2. Don't represent distinctions that you don't need.

   Fall leaves fall and spring leaves spring.

3. Don't do once what you can avoid altogether.

   The can holds the water. ("can": AUX, V, N)

Earley Algorithm: Top-Down Chart Parser

For all S rules of the form $S \rightarrow X_1 \ldots X_k$, add a (top-down) edge from 1 to 1 labeled: $S \rightarrow \circ X_1 \ldots X_k$.

Do until there is no input left:

1. If the agenda is empty, look up word categories for next word, add to agenda.
2. Select a constituent from the agenda: constituent C from p_1 to p_2.
3. Using the (bottom-up) edge extension algorithm, combine C with every active edge on the chart (adding C to chart as well). Add any new constituents to the agenda.
4. For any active edges created in Step 3, add them to the chart using the top-down edge introduction algorithm.

Top-down edge introduction.

To add an edge $S \rightarrow C_1 \ldots \circ C_i \ldots C_n$ ending at position j:

For each rule in the grammar of form $C_i \rightarrow X_1 \ldots X_k$, recursively add the new edge $C_i \rightarrow \circ X_1 \ldots X_k$ from j to j.

Grammar and Lexicon

Grammar	Lexicon
1. S → NP VP	the: ART
2. NP → ART ADJ N	large: ADJ
3. NP → ART N	can: N, AUX, V
4. NP → ADJ N	hold: N, V
5. VP → AUX VP	water: N, V
6. VP → V NP	

Sentence: $_1$ The $_2$ large $_3$ can $_4$ can $_5$ hold $_6$ water $_7$