Parsing with CFGs

L545

Dept. of Linguistics, Indiana University
Spring 2013

Parsing with CFGs: Overview

Input: a string
Output: a (single) parse tree

- A useful step in the process of obtaining meaning
- We can view the problem as searching through all possible parses (tree structures) to find the right one

Strategies:

- top-down (goal-directed) vs. bottom-up (data-directed)
- depth-first vs. breadth-first
- left-corner parsing (adding bottom-up to top-down)
- chart parsing (saving partial results)

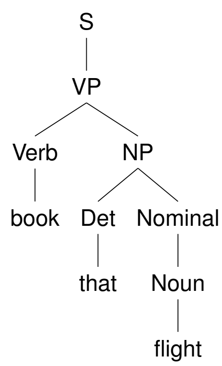
Parsers and criteria to evaluate them

- Function of a parser:
  - grammar + string → analysis trees
- Main criteria for evaluating parsers:
  - Correctness: for every grammar and for every string, every analysis returned by parser is an actual analysis
    - Correctness w.r.t. our target language is thus dependent upon the grammar we give the parser
  - Completeness: for every grammar and for every string, every correct analysis is found by the parser
    - For large grammars, this may not be practical, and for some situations, we may want only one analysis
  - Efficiency: storing partial parses is essential in being efficient (to be explained)

Example grammar and desired tree

Sentence: Book that flight.

- S → NP VP
- S → Aux NP VP
- S → VP
- NP → Det Nominal
- Nominal → Noun
- Nominal → Noun Nominal
- Nominal → Nominal PP
- NP → Proper-Noun
- VP → Verb
- VP → Verb NP



Direction of processing I
Top-down

Goal-driven processing is top-down:

- Start with the start symbol
- Derive sentential forms
  - If the string is among the sentences derived this way, it is part of the language

Problem: Left-recursive rules (e.g., NP → NP PP) can give rise to infinite hypotheses

- Plus, we can expand non-terminals which cannot lead to the existing input
- No tree takes the properties of the lexical items into account until the last stage

How are alternatives explored?
I. Depth-first

At every choice point: Pursue a single alternative completely before trying another alternative

- State of affairs at the choice points needs to be remembered. Choices can be discarded after unsuccessful exploration.
- Depth-first search is not necessarily complete.

Problem for top-down, left-to-right, depth-first processing:

- left-recursion
  For example, a rule like N' → N' PP leads to non-termination.

How are alternatives explored?

II. Breadth-first

At every choice point: Pursue every alternative for one step at a time

- Requires serious bookkeeping since each alternative computation needs to be remembered at the same time.
- Search is guaranteed to be complete.

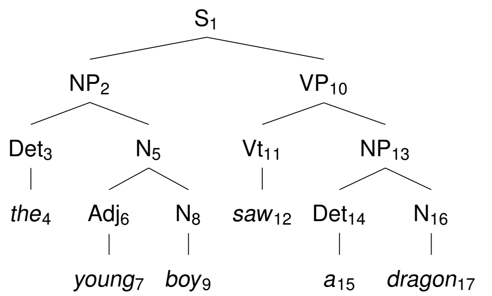
An example grammar

Lexicon:
Vt → *saw*
Det → *the*
Det → *a*
N → *dragon*
N → *boy*
Adj → *young*

Syntactic rules:
S → NP VP
VP → Vt NP
NP → Det N
N → Adj N

Top-down, left-right, depth-first tree traversal

S → NP VP
VP → Vt NP
NP → Det N
N → Adj N
Vt → saw
Det → the
Det → a
N → dragon
N → boy
Adj → young



A walk-through

Goal	Input	Action
S	the young boy saw the dragon	expand S
NP VP	the young boy saw the dragon	expand NP
Det N VP	the young boy saw the dragon	expand Det
the N VP	the young boy saw the dragon	consume *the*
N VP	young boy saw the dragon	expand N
dragon VP	young boy saw the dragon	fail with *dragon*
boy VP	young boy saw the dragon	fail with *boy*; (re)expand N
Adj N VP	young boy saw the dragon	expand Adj
young N VP	young boy saw the dragon	consume *young*
N VP	boy saw the dragon	expand N

A walk-through (cont.)

Goal	Input	Action
dragon VP	boy saw the dragon	fail with *dragon*
boy VP	boy saw the dragon	consume *boy*
VP	saw the dragon	expand VP
Vt NP	saw the dragon	expand Vt
saw NP	saw the dragon	consume *saw*
NP	the dragon	expand NP
Det N	the dragon	expand Det
the N	the dragon	consume *the*
N	dragon	expand N
dragon	dragon	consume *dragon*
<empty>	<empty>	SUCCESS!

Remaining choices

There are still some choices that have to be made:

1. Which leaf node should be expanded first?
   - Left-to-right strategy moves through the leaf nodes in a left-to-right fashion
2. Which rule should be applied first for multiple rules with same LHS?
   - Can just use the textual order of rules from the grammar
   - There may be reasons to take rules in a particular order (e.g., probabilities)

Parsing with an agenda

Search states are kept in an agenda

- Search states consist of partial trees and a pointer to the next input word in the sentence

Based on what we've seen before, apply the next item on the agenda to the current tree

- Add new items to the agenda, based on the rules in the grammar which can expand at the (leftmost) node
  - We maintain the depth-first strategy by adding new hypotheses (rules) to the front of the agenda
  - If we added them to the back, we would have a breadth-first strategy

Direction of processing II

Bottom-up

Data-driven processing is bottom-up:

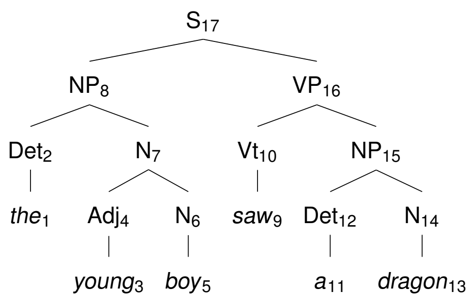
- Start with the sentence.
- For each substring, find a grammar rule which covers it.
- If you finish with a sentence, it is grammatical.

Problem: Epsilon rules ($N \rightarrow \epsilon$) allow us to hypothesize empty categories anywhere in the sentence.

- Also, while any parse in progress is tied to the input, many may not lead to an S!

Bottom-up, left-right, depth-first tree traversal

S → NP VP	Det → the
VP → Vt NP	Det → a
NP → Det N	N → dragon
N → Adj N	N → boy
Vt → saw	Adj → young



A walk-through

Analysis	Input	Action
<empty>	the young boy saw the dragon	shift *the*
the	young boy saw the dragon	reduce *the* to Det
Det	young boy saw the dragon	shift *young* after failing to reduce Det
Det young	boy saw the dragon	reduce *young* to Adj after failing to reduce Det *young*
Det Adj	boy saw the dragon	shift *boy*
Det Adj boy	saw the dragon	reduce *boy* to N
Det Adj N	saw the dragon	reduce Adj N to N
Det N	saw the dragon	reduce Det N to NP
NP	saw the dragon	shift *saw*

A walk-through (cont.)

Analysis	Input	Action
NP saw	the dragon	reduce *saw* to Vt
NP Vt	the dragon	shift *the*
NP Vt the	dragon	reduce *the* to Det
NP Vt Det	dragon	shift *dragon*
NP Vt Det dragon	<empty>	reduce *dragon* to N
NP Vt Det N	<empty>	reduce Det N to NP
NP Vt NP	<empty>	reduce Vt NP to VP
NP VP	<empty>	reduce NP VP to S
S	<empty>	SUCCESS!

Left-corner parsing

Motivation:

- Both pure top-down & bottom-up approaches are inefficient
- The correct top-down parse has to be consistent with the left-most word of the input

Left-corner parsing: a way of using bottom-up constraints as part of a top-down strategy.

- Left-corner rule:
  - expand a node with a grammar rule only if the current input can serve as the left corner from this rule
- Left-corner from a rule: first word along the left edge of a derivation from the rule

Put the left-corners into a table, which then guide parsing

Grammar with left-corners

Lexicon:	Syntactic rules:	Left corners:
Vt → *saw*	S → NP VP	S ⇒ Det
Det → *the*	VP → Vt NP	VP ⇒ Vt
Det → *a*	NP → Det N	NP ⇒ Det
N → *dragon*	N → Adj N	N ⇒ Adj
N → *boy*		
Adj → *young*		

Left corner parsing example

Consider again *book that flight*, with these rules:

S → NP VP	Nom. → Noun	VP → Verb
S → Aux NP VP	Nom. → Noun Nom.	VP → Verb NP
S → VP	Nom. → Nom. PP	
NP → Det Nom.	NP → Proper-Noun	

With an ambiguous word like *book*, left corners tell us the Noun reading is ruled out—it cannot start an S

S ⇒ Aux	S ⇒ Verb	VP ⇒ Verb
S ⇒ Det	NP ⇒ Det	
S ⇒ PropN	NP ⇒ PropN	

Moving top-down, we hypothesize S → NP VP, but the NP's left-corner is incompatible with any category of *book*

- Thus, no NP expansions are considered

Chart parsing

Problem: Inefficiency of recomputing subresults

Two example sentences and their potential analysis:

(1) He [gave [the young cat] [to Bill]].

(2) He [gave [the young cat] [some milk]].

The corresponding grammar rules:

- VP → $V_{ditrans}$ NP PP_{to}
- VP → $V_{ditrans}$ NP NP

Regardless of final sentence analysis, the object NP (*the young cat*) will have the same analysis

⇒ No need to analyze it twice

Solution: Chart Parsing (Memoization)

- Store intermediate results:
  a) completely analyzed constituents: **well-formed substring table** or **(passive) chart**
  b) partial and complete analyses: **(active) chart**
- In other words, instead of recalculating that *the young cat* is an NP, we'll store that information
  - Dynamic programming: never go backwards
- All intermediate results need to be stored for completeness.
- All possible solutions are explored in parallel.

Cocke Younger Kasami (CYK) Algorithm

- Grammar has to be in Chomsky Normal Form (CNF):
  - 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 $\cdot_6$

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) = $\{A \mid A \Rightarrow^* w_{i+1} \ldots w_j\}$
- 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:

$\cdot_0$ the $\cdot_1$ young $\cdot_2$ boy $\cdot_3$ saw $\cdot_4$ the $\cdot_5$ 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
- This algorithm is used to:
  - add top-down guidance (only use rules derivable from start-symbol), but avoid left-recursion problem
  - 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:

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

syntactic_chart_fill(i,j)

- Idea: Perform all reduction steps using syntactic rules s.t. 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
              $chart(i,j) := chart(i,j) \cup \{A\}$.

The Complete CYK Algorithm

Input: start category S and input *string*

n := length(*string*)

for j := 1 to n
    $chart(j-1,j) := \{X \mid X \rightarrow word_j \in P\}$
    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
                    $chart(i,j) := chart(i,j) \cup \{A\}$

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 CYK to Earley

- CKY algorithm:
  - explores all analyses in parallel
  - bottom-up
  - stores complete subresults
- desiderata:
  - add top-down guidance (to only use rules derivable from start-symbol), but avoid left-recursion problem of top-down parsing
  - store partial analyses (useful for rules right-hand sides longer than 2)
- Idea: also store partial results, so that the chart contains
  - passive items: complete results
  - active items: partial results

Representing active chart items

- well-formed substring entry:
  `chart(i,j,A)`: from `i` to `j` there is a constituent of category `A`
- More elaborate data structure needed to store partial results:
  - rule considered + how far processing has succeeded
  - dotted rule:
    $_i[A \rightarrow \alpha \bullet_j \beta]$
- active chart entry:
  `chart(i,j,state(A,`β`))` Note: α is not represented
  A (incompletely) goes from i to j and can be completed by finding β

Dotted rule examples

- A dotted rule represents a state in processing a rule.
- Each dotted rule is a hypothesis:

	We found a *vp* if we still find
vp $\rightarrow \bullet$ *v-ditr np pp-to*	a *v-ditr*, a *np*, and a *pp-to*
vp $\rightarrow$ *v-ditr* $\bullet$ *np pp-to*	a *np* and a *pp-to*
vp $\rightarrow$ *v-ditr np* $\bullet$ *pp-to*	a *pp-to*
vp $\rightarrow$ *v-ditr np pp-to* $\bullet$	nothing

  - The first three are **active items** (or **active edges**)
  - The last one is a **passive item/edge**

The three actions in Earley's algorithm

In $_i[A \rightarrow \alpha \bullet_j B\beta]$ we call B the *active constituent*.

- **Prediction:** Search all rules realizing the active constituent.
- **Scanning**: Scan over each word in the input string.
- **Completion:** Combine an active edge with each passive edge covering its active constituent.

Success state: $_0[start \rightarrow s \bullet_n]$

A closer look at the three actions Prediction

Prediction: for each $_i[A \rightarrow \alpha \bullet_j B\,\beta]$ in chart
for each $B \rightarrow \gamma$ in rules
add $_j[B \rightarrow \bullet_j \gamma]$ to chart

Prediction is the task of saying what kinds of input we expect to see

- Add a rule to the chart saying that we have not seen γ, but when we do, it will form a B
- The rule covers no input, so it goes from j to j

Such rules provide the top-down aspect of the algorithm

A closer look at the three actions Scanning

Scanning: let $w_1 \ldots w_j \ldots w_n$ be the input string
for each $_i[A \rightarrow \alpha \bullet_{j-1} w_j\,\beta]$ in chart
add $_i[A \rightarrow \alpha\, w_j \bullet_j \beta]$ to chart

Scanning reads in lexical items

- We add a dotted rule indicating that a word has been seen between $j-1$ and j
- Such a completed dotted rule can be used to complete other dotted rules

These rules provide the bottom-up component to the algorithm

A closer look at the three actions Completion

Completion (fundamental rule of chart parsing):

for each $_i[A \rightarrow \alpha \bullet_k B\,\beta]$ and $_k[B \rightarrow \gamma \bullet_j]$ in chart
add $_i[A \rightarrow \alpha\, B \bullet_j \beta]$ to chart

Completion combines two rules in order to move the dot, i.e., indicate that something has been seen

- A rule covering B has been seen, so any rule A which refers to B in its RHS moves the dot
- Instead of spanning from i to k, A now spans from i to j, which is where B ended

Once the dot is moved, the rule will not be created again

Eliminating scanning

Scanning: for each $_i[A \rightarrow \alpha \bullet_{j-1} w_j\,\beta]$ in chart
add $_i[A \rightarrow \alpha\, w_j \bullet_j \beta]$ to chart

Completion:
for each $_i[A \rightarrow \alpha \bullet_k B\,\beta]$ and $_k[B \rightarrow \gamma \bullet_j]$ in chart
add $_i[A \rightarrow \alpha\, B \bullet_j \beta]$ to chart

Observation: Scanning = completion + words as passive edges. One can thus simplify scanning to adding a passive edge for each word:

for each w_j in $w_1 \ldots w_n$
add $_{j-1}[w_j \rightarrow \bullet_j]$ to chart

Earley's algorithm without scanning

General setup:
apply prediction and completion to every item added to chart

Start: add $_0[start \rightarrow \bullet_0\, s]$ to chart

for each w_j in $w_1 \ldots w_n$
add $_{j-1}[w_j \rightarrow \bullet_j]$ to chart

Success state: $_0[start \rightarrow s \bullet_n]$

A tiny example grammar

Lexicon:

vp	→	left
det	→	the
n	→	boy
n	→	girl

Syntactic rules:

s	→	np vp
np	→	det n

An example run

Parsing with CFGs
Direction of processing
Top-down
Bottom-up
Left-corner parsing
Chart parsing
CYK
Earley

start	1. $_0$[start → $\bullet_0$ s]
predict from 1	2. $_0$[s → $\bullet_0$ np vp]
predict from 2	3. $_0$[np → $\bullet_0$ det n]
predict from 3	4. $_0$[det → $\bullet_0$ the]
scan "the"	5. $_0$[the → $\bullet_1$]
complete 4 with 5	6. $_0$[det → the $\bullet_1$]
complete 3 with 6	7. $_0$[np → det $\bullet_1$ n]
predict from 7	8. $_1$[n → $\bullet_1$ boy]
predict from 7	9. $_1$[n → $\bullet_1$ girl]
scan "boy"	10. $_1$[boy → $\bullet_2$]
complete 8 with 10	11. $_1$[n → boy $\bullet_2$]
complete 7 with 11	12. $_0$[np → det n $\bullet_2$]
complete 2 with 12	13. $_0$[s → np $\bullet_2$ vp]
predict from 13	14. $_2$[vp → $\bullet_2$ left]
scan "left"	15. $_2$[left → $\bullet_3$]
complete 14 with 15	16. $_2$[vp → left $\bullet_3$]
complete 13 with 16	17. $_0$[s → np vp $\bullet_3$]
complete 1 with 17	18. $_0$[start → s$\bullet_3$]

Improving the efficiency of lexical access

Parsing with CFGs
Direction of processing
Top-down
Bottom-up
Left-corner parsing
Chart parsing
CYK
Earley

- In the setup just described:
  - Words are stored as passive items so that prediction is used for preterminal categories
  - Set of predicted words for a preterminal can be huge
- If each word in the grammar is introduced by a preterminal rule *cat* → *word*, one can add a **passive item for each preterminal category** which can dominate the word instead of for the word itself
- What needs to be done:
  - syntactically distinguish syntactic rules from rules with preterminals on the left-hand side, i.e. lexical entries.
  - modify scanning to take lexical entries into account

Earley parsing

Parsing with CFGs
Direction of processing
Top-down
Bottom-up
Left-corner parsing
Chart parsing
CYK
Earley

The Earley algorithm is efficient, running in polynomial time.

- Technically, however, it is a recognizer, not a parser

To make it a parser, each state needs to be augmented with a pointer to the states that its rule covers

- For example, VP points to state where V was completed and state where NP was completed
- Also true of the CKY algorithm: pointers need to be added to make it a parser