Lecture 16:
The CKY parsing algorithm

Julia Hockenmaier
juliahm@illinois.edu
3324 Siebel Center

Survey results (Thank you!)
23 responses so far: 56% undergrads, 44% grads
Lectures are
– interesting/useful: 70%; boring/useless: 30%
– clear: 13%, confusing 35% (talk to us!!!!)
The level of the class is
– a bit too easy: 17%, just right: 65%,
  a bit too difficult: 9%, much too difficult: 9% (talk to us!!)
Time required is
– a bit too little: 9%, just right: 48%,
  a bit too much: 39%, much too much: 4%
Material is presented
– much too fast: 13%, too fast 30%, just right 30%
– a bit too slow 17%, much too slow: 9%
Grading is fair: 91%, unfair: 9%

Last lecture’s key concepts
Natural language syntax
  Constituents
  Dependencies
  Context-free grammar
  Arguments and modifiers
  Recursion in natural language

Today’s class
Parsing with CFGs:
The CKY (Cocke Kasami Younger) algorithm
An example CFG

- DT → {the, a}
- N → {ball, garden, house, sushi}
- P → {in, behind, with}
- NP → DT N
- NP → NP PP
- PP → P NP

N: noun
P: preposition
NP: “noun phrase”
PP: “prepositional phrase”

Reminder: Context-free grammars

A CFG is a 4-tuple \( \langle N, \Sigma, R, S \rangle \) consisting of:

- A set of nonterminals \( N \)
  (e.g. \( N = \{ S, \text{NP}, \text{VP}, \text{PP}, \text{Noun}, \text{Verb}, \ldots \} \))
- A set of terminals \( \Sigma \)
  (e.g. \( \Sigma = \{ I, \text{you}, \text{he}, \text{eat}, \text{drink}, \text{sushi}, \text{ball}, \} \))
- A set of rules \( R \)
  \( R \subseteq \{ \lambda \to \beta \text{ with left-hand-side (LHS) } \lambda \in N \text{ and right-hand-side (RHS) } \beta \in (N \cup \Sigma)^* \} \)
- A start symbol \( S \in N \)

Chomsky Normal Form

The right-hand side of a standard CFG can have an arbitrary number of symbols (terminals and nonterminals):

- VP → ADV eat NP

A CFG in Chomsky Normal Form (CNF) allows only two kinds of right-hand sides:

- Two nonterminals: VP → ADV VP
- One terminal: VP → eat

Any CFG can be transformed into an equivalent CNF:

- VP → ADVP VP₁
- VP₁ → VP₂ NP
- VP₂ → eat

A note about \( \varepsilon \)-productions

Formally, context-free grammars are allowed to have empty productions (\( \varepsilon \) = the empty string):

- VP → V NP
- NP → DT Noun
- NP → \( \varepsilon \)

These can always be eliminated without changing the language generated by the grammar:

- VP → V NP
- NP → DT Noun
- NP → \( \varepsilon \)

becomes

- VP → V NP
- VP → V \varepsilon
- NP → DT Noun

which in turn becomes

- VP → V NP
- VP → V
- NP → DT Noun

We will assume that our grammars don’t have \( \varepsilon \)-productions
CKY chart parsing algorithm

Bottom-up parsing:
start with the words
Dynamic programming:
save the results in a table/chart
re-use these results in finding larger constituents

Complexity: $O(n^3|G|)$
$n$: length of string, $|G|$: size of grammar

Presumes a CFG in Chomsky Normal Form:
Rules are all either $A \rightarrow BC$ or $A \rightarrow a$
(with $A,B,C$ nonterminals and $a$ a terminal)

CKY algorithm

1. Create the chart
(an $n \times n$ upper triangular matrix for an sentence with $n$ words)
– Each cell chart[i][j] corresponds to the substring $w(i)...w(j)$
2. Initialize the chart (fill the diagonal cells chart[i][i]):
For all rules $X \rightarrow w(i)$, add an entry $X$ to chart[i][i]
3. Fill in the chart:
Fill in all cells chart[i][i+1], then chart[i][i+2], ..., until you reach chart[1][n] (the top right corner of the chart)
– To fill chart[i][j], consider all binary splits $w(i)...w(k)w(k+1)...w(j)$
– If the grammar has a rule $X \rightarrow YZ$, chart[i][k] contains a $Y$ and chart[k+1][j] contains a $Z$, add an $X$ to chart[i][j] with two backpointers to the $Y$ in chart[i][k] and the $Z$ in chart[k+1][j]
4. Extract the parse trees from the S in chart[1][n].

The CKY parsing algorithm

To recover the parse tree, each entry needs pairs of backpointers.

We eat sushi
The CKY parsing algorithm

Each cell may have one entry for each nonterminal.

We buy drinks with milk

What are the terminals in NLP?

Are the “terminals”: words or POS tags?

For toy examples (e.g. on slides), it’s typically the words

With POS-tagged input, we may either treat the POS tags as the terminals, or we assume that the unary rules in our grammar are of the form

POS-tag → word

(so POS tags are the only nonterminals that can be rewritten as words; some people call POS tags “preterminals”)
Additional unary rules

In practice, we may allow other unary rules, e.g.
   NP → Noun
(where Noun is also a nonterminal)

In that case, we apply all unary rules to the entries in chart[i][j] after we’ve checked all binary splits
(chart[i][k], chart[k+1][j])

Unary rules are fine as long as there are no “loops” that could lead to an infinite chain of unary productions, e.g.:
   X → Y and Y → X
   or: X → Y and Y → Z and Z → X

Exercise: CKY parser

I eat sushi with chopsticks with you

S  → NP  VP
NP → NP  PP
NP → sushi
NP → I
NP → chopsticks
NP → you
VP → VP  PP
VP → Verb  NP
Verb → eat
PP → Prep  NP
Prep → with

How do you count the number of parse trees for a sentence?

1. For each pair of backpointers (e.g. VP → V NP): multiply #trees of children
   trees(VP_{VP → V NP}) = trees(V) × trees(NP)

2. For each list of pairs of backpointers (e.g. VP → V NP and VP → VP PP): sum #trees
   trees(VP) = trees(VP_{VP → V NP}) + trees(VP_{VP → VP PP})
Cocke Kasami Younger (1)

\[ \text{ckyParse}(n): \]
\[ \text{initChart}(n): \]
\[ \text{for } i = 1..n: \]
\[ \text{initCell}(i,i) \]
\[ \text{initCell}(i,i): \]
\[ \text{for } c \text{ in lex(word[i]):} \]
\[ \text{addToCell(cell[i][i], c, null, null)} \]
\[ \text{addToCell(Parent, cell, Left, Right)} \]
\[ \text{if (cell.hasEntry(Parent)):} \]
\[ P = \text{cell.getEntry(Parent)} \]
\[ P . \text{addBackpointers(Left, Right)} \]
\[ \text{else cell.addEntry(Parent, Left, Right)} \]

\[ \text{fillChart}(n): \]
\[ \text{for span = 1..n-1:} \]
\[ \text{for } i = 1..n-span: \]
\[ \text{fillCell(i,i+span)} \]

\[ \text{fillCell}(i,j): \]
\[ \text{for } k = i..j-1: \]
\[ \text{combineCells}(i, k, j) \]

\[ \text{combineCells}(i,k,j): \]
\[ \text{for } Y \text{ in cell}[i][k]: \]
\[ \text{for } Z \text{ in cell}[k+1][j]: \]
\[ \text{for } X \text{ in Nonterminals:} \]
\[ \text{if } X \rightarrow Y\,Z \text{ in Rules:} \]
\[ \text{addToCell(cell}[i][j],X, Y, Z) \]

\[ \text{combineCells}(i, k, j): \]
\[ \text{for } Y \text{ in cell}[i][k]: \]
\[ \text{for } Z \text{ in cell}[k+1][j]: \]
\[ \text{for } X \text{ in Nonterminals:} \]
\[ \text{if } X \rightarrow Y\,Z \text{ in Rules:} \]
\[ \text{addToCell(cell}[i][j],X, Y, Z) \]