UPPSALA UNIVERSITET

CKY discussion session

Syntactic parsing

2023

Sara Stymne
Department of Linguistics and Philology

Mostly based on slides from Marco Kuhlmann

UPPSALA UNIVERSITET

CNF conversion

Restrictions

- The original CKY algorithm can only handle rules that are at most binary:
  $C \rightarrow w_i\,,\ C \rightarrow C_1\ C_2\,.$
- It can easily be extended to also handle unit productions:
  $C \rightarrow w_i\,,\quad C \rightarrow C_1\,,\quad C \rightarrow C_1\ C_2\,.$
- This restriction is not a problem theoretically, but requires preprocessing (binarization) and postprocessing (debinarization).
- A parsing algorithm that does away with this restriction is Earley's algorithm (Lecture 5 and J&M 13.4.2).

UPPSALA UNIVERSITET

Treebank CNF conversion (1)

Probably easiest to solve by a recursive function. XXX represents either a list or a string

A tree is represented as a list of subtrees, e.g.

[S [NP [PRON they]] [VP [V like] [NP [N snow]]]]

```
List contains two strings
e.g.:  ["IN", "as"]
    return list

List contains two items, string and list
e.g. : ["NP" ["PRP", XXX]]
    Contract the two grammar symbols, and remove one list
    Apply cnf-method to the resulting tree
      return cnf(["NP+PRP", XXX])
List contains three symbols, string, list, list
e.g. ["NP", ["DT", XXX], ["NNS", XXX]]
    Keep as it is, and apply cnf-method to the two lists
      return [”NP”, cnf(["DT", XXX]), cnf(["NNS", XXX])]
```

UPPSALA UNIVERSITET

Treebank CNF conversion (2)

```
List contains more than three symbols, string, list,
list, list, ...
e.g. ["S", ["NP", XXX], ["VP", XXX], [".", XXX]]
    Keep first two items, create an extra list with
    new label to which you give a "new" label.
    Apply cnf to the resulting tree
      return cnf(["S", ["NP", XXX],
          ["new-name",  ["VP", XXX], [".", XXX]]])

    # think about the naming and markovization!

List contains something else:
    Something has gone wrong!
```

UPPSALA UNIVERSITET

CNF Conversion task

- Note a small change in the assignment from previous years:
  - Instead of changing the list ”in-place”, you are now required to return the new list.
  - This change was made as a simplification, since many students previously struggled with the in-place conversion
    - Please disregard any mention of in-place conversion that are still in the recordings

UPPSALA
UNIVERSITET

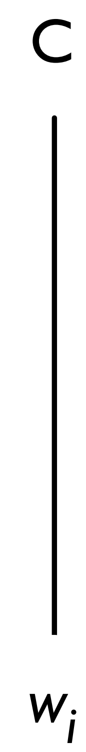
The CKY algorithm

UPPSALA UNIVERSITET

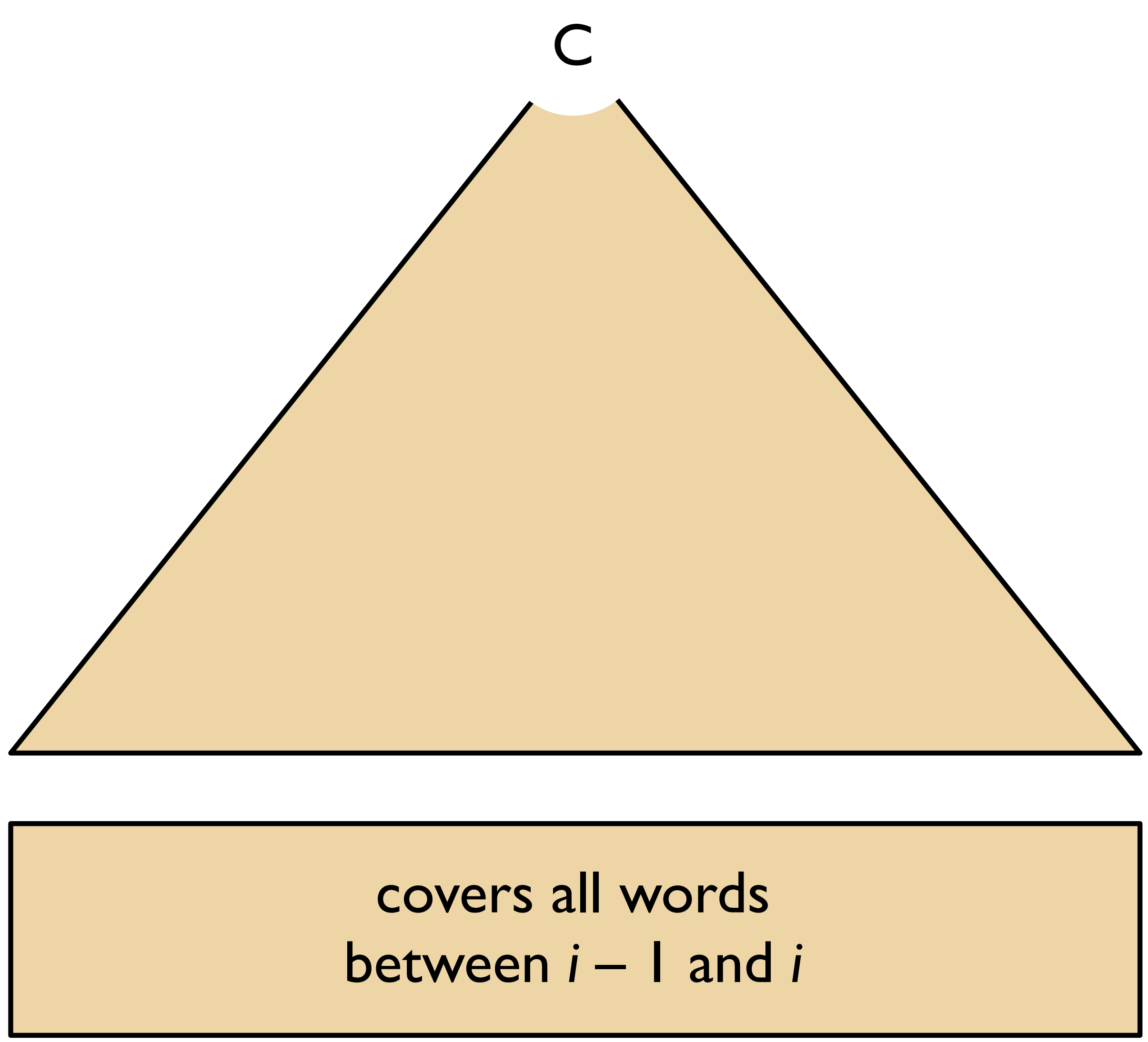
Overview of the CKY algorithm

- The CKY algorithm is an efficient bottom-up parsing algorithm for context-free grammars.
- It was discovered at least three (!) times and named after Cocke, Kasami, and Younger.
- It is one of the most important and most used parsing algorithms.

Recognizing small trees

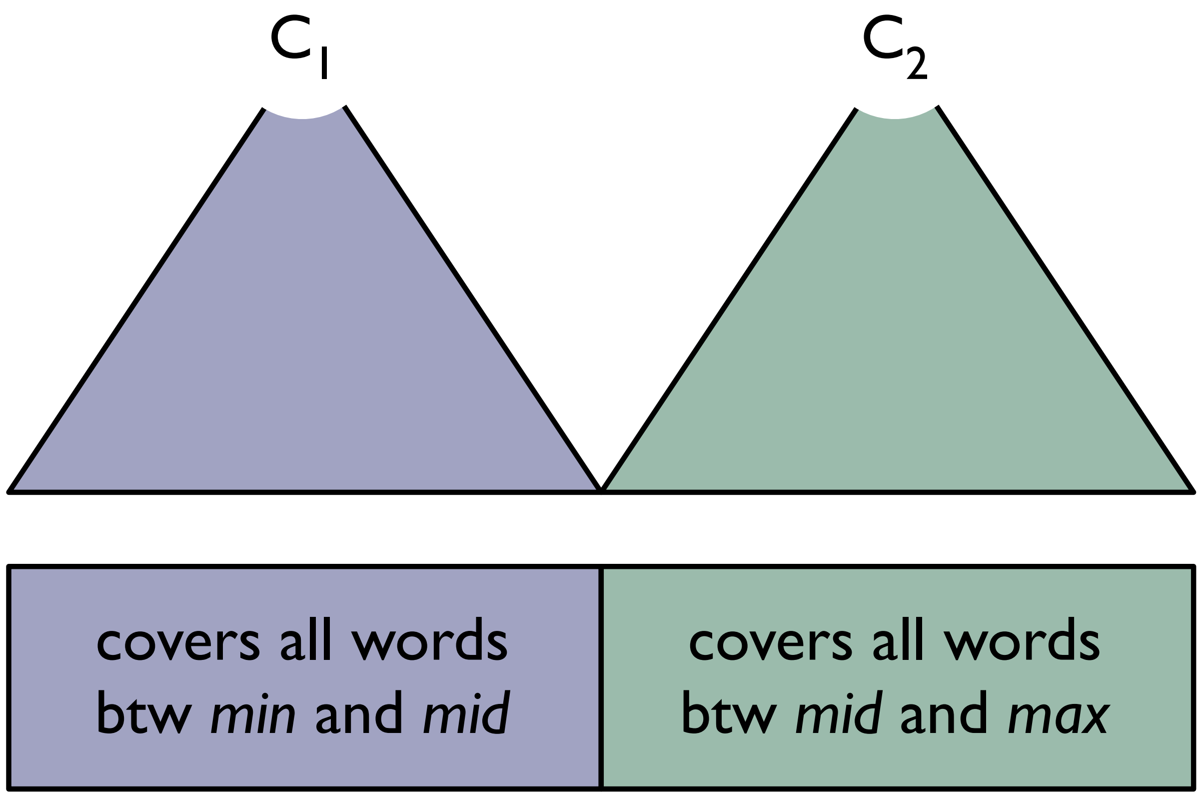


Recognizing small trees

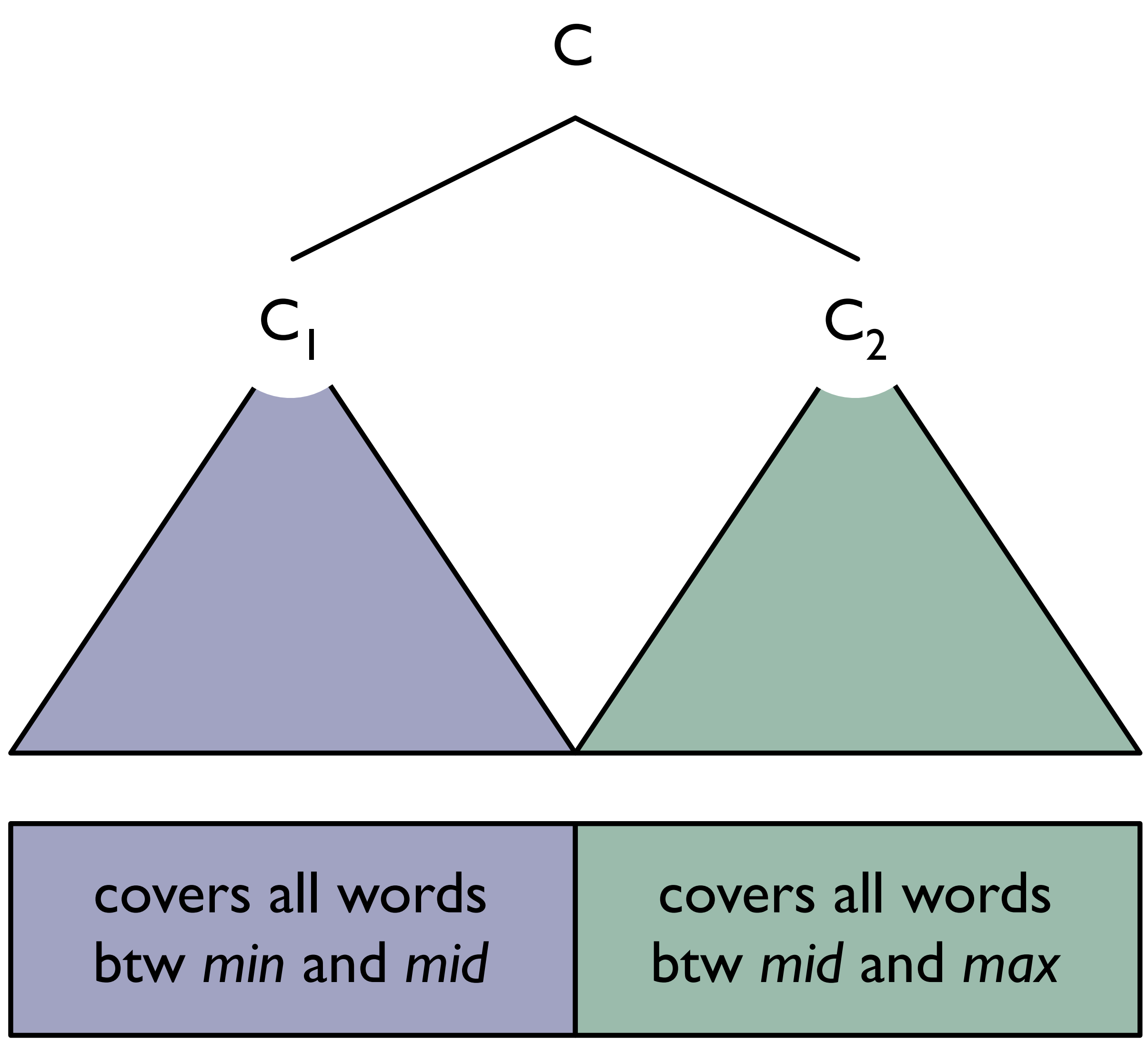


Recognizing big trees

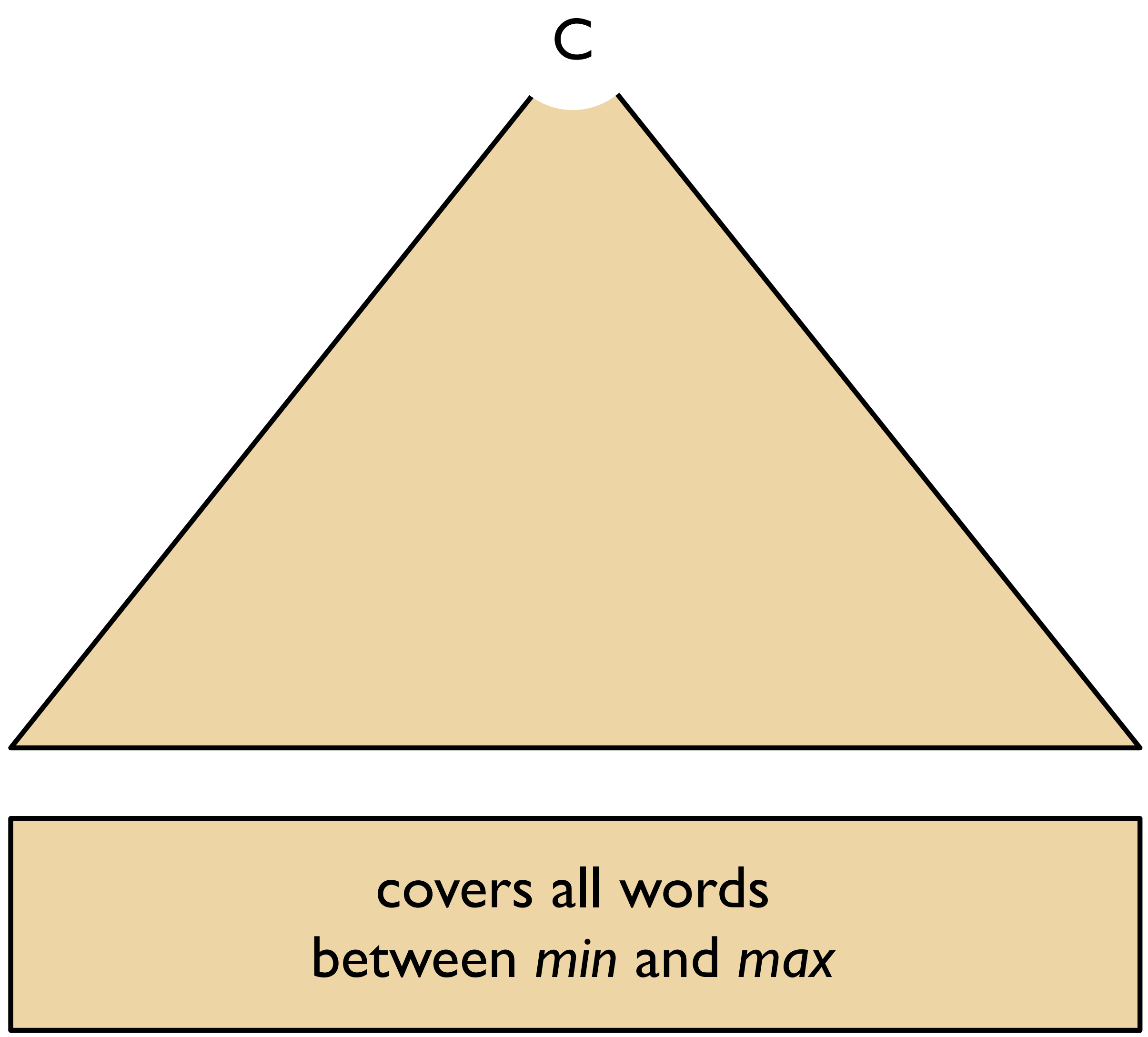
$C \rightarrow C_1\ C_2$



Recognizing big trees



Recognizing big trees



Questions, CKY recognition

- How do we know that we have recognized that the input sequence is grammatical?
- How do we need to extend this reasoning in the presence of unary rules: $C \rightarrow C_1$?

Questions

- What is the signature of a parse tree for the complete sentence?
- How many different signatures are there?
- Can you relate the runtime of the parsing algorithm to the number of signatures?

Questions

- What is the signature of a parse tree for the complete sentence?
  - [0, n, S]
- How many different signatures are there?
  - n^2 * G
- Can you relate the runtime of the parsing algorithm to the number of signatures?
  - n^3 * G

UPPSALA UNIVERSITET

Implementation CKY recognizer

Implementation

Preterminal rules

```
for each wᵢ from left to right
  for each preterminal rule C -> wᵢ
    chart[i - 1][i][C] = true
```

Implementation

Binary rules

```
for each max from 2 to n
  for each min from max - 2 down to 0
    for each syntactic category C
      for each binary rule C -> C₁ C₂
        for each mid from min + 1 to max - 1
          if chart[min][mid][C₁] and chart[mid][max][C₂] then
            chart[min][max][C] = true
```

Questions, CKY recognizer

- In what way is this algorithm bottom–up?
- Why is that property of the algorithm important?
- How do we need to extend the code if we wish to handle unary rules $C \rightarrow C_l$?
  - Why would we want to do that?

UPPSALA UNIVERSITET

Unary rules

new bounds!

```
for each max from 1 to n
  for each min from max - 1 down to 0
    // First, try all binary rules as before.
    ...
    // Then, try all unary rules.
    for each syntactic category C
      for each unary rule C -> C1
        if chart[min][max][C1] then
          chart[min][max][C] = true
```

UPPSALA UNIVERSITET

Question, unary rules

This is not quite right.

Why, and how could we fix the problem?

UPPSALA
UNIVERSITET

CKY parser

Idea

- For trees built using preterminal rules:
  Find a most probable rule. (apply all rules!)
- For trees built using binary rules:
  Find a binary rule *r* and a split point *mid* such that $p(r) \times p(t_l) \times p(t_2)$ is maximal, where
  t_l is a most probable left subtree and
  t_2 is a most probable right subtree.

Preterminal rules

```
for each wi from left to right
  for each preterminal rule C -> wi
    chart[i - 1][i][C] = p(C -> wi)
```

Binary rules

```
for each max from 2 to n
  for each min from max - 2 down to 0
    for each syntactic category C
      double best = undefined
      for each binary rule C -> C₁ C₂
        for each mid from min + 1 to max - 1
          double t₁ = chart[min][mid][C₁]
          double t₂ = chart[mid][max][C₂]
          double candidate = t₁ * t₂ * p(C -> C₁ C₂)
          if candidate > best then
            best = candidate
      chart[min][max][C] = best
```

Question

How should we treat unary rules?

Backpointers

- When we find a new best parse tree, we want to remember how we built it.
- For each element t = $chart[min][max][C]$, we also store backpointers to those elements from which t was built.
- Besides the ordinary chart of floats, we also have a backpointer chart

Preterminal rules

```
for each wi from left to right
  for each preterminal rule C -> wi
    chart[i - 1][i][C] = p(C -> wi)
    backpointerChart[i-1][i][C] = (C, wi, i, i-1)
```

Backpointers

```
double best = undefined
Backpointer backpointer = undefined
...
if candidate > best then
  best = candidate
  // We found a better tree; update the backpointer!
  backpointer = (C, C1, C2, min, mid, max)
...
chart[min][max][C] = best
backpointerChart[min][max][C] = backpointer
```

Implementation

Backtrace

Convenient to use recursion to retrieve the tree!

```
# assume backppointers are tuples:

# Preterminal: (C, w, min, max)

# Binary: (C, C1, C2, min, mid, max)

backtrace(bp, bpChart):

    if length(bp) == 4:  #preterminal rule

        return tree for C, w

    else if length(bp) == 6 #binary rule

        return tree for C, backtrace(left subtree), backtrace(right
        subtree)
```

Implementation ideas, Python

```
#  defaultdict is a suitable datastructure for charts!
# Index the defaultdicts with a tuple (min, max, cat)
    pi = defaultdict(float)
    bp = defaultdict(tuple)
#  Recognize all parse trees built with with preterminal rules.
#  Recognize all parse trees built with binary rules.

#  ″S″ is not always the top category, the below is a simplification
return backtrace(bp[0, n, "S"], bp);
```

Assignment 1: Lab sessions

- First lab session:
  - Will be moved:
    - From: this Wednesday (Jan. 25)
    - To: Monday, Jan. 30, 10-12
- Second lab session:
  - Not scheduled at all by mistake
  - Will be in the afternoon on Feb. 8 (now scheduled for seminar 1, which will be before lunch)
- TimeEdit still to be updated by these changes!

Assignment 1: CKY parsing

- Tips:
  - During development: use print statements to make sure your code does what you think it should
  - Use a small test set, and possibly a small grammar during development. The parser is slow
  - Start on the assignment now! Do not leave it until the last week!
  - Come to the lab sessions and ask questions!
  - You can also contact me for help!

Coming up

- Monday 10-12:
  - First lab session for CKY assignment
- Next theme:
  - Treebanks and Earley's algorithm (JM: 17.3)
    - Recorded lectures and exercises will be available in Studium
    - Lecture Monday 13-15
  - Seminar 1: Wednesday February 8
    - Groups+times will be posted on the web page