



UPPSALA
UNIVERSITET

CKY discussion session

Syntactic parsing

2024

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:
 $C \rightarrow w_i, C \rightarrow C_1, C \rightarrow C_1 C_2.$
- 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 6).



Treebank CNF conversion (I)

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. ["S", ["NP", XXX], ["VP", XXX]]

Keep as it is, and apply cnf-method to the two lists

```
return ["S", cnf(["NP", XXX]), cnf(["VP", XXX])]
```



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!



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 – recognition



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.



UPPSALA
UNIVERSITET

Recognition

Recognizing small trees

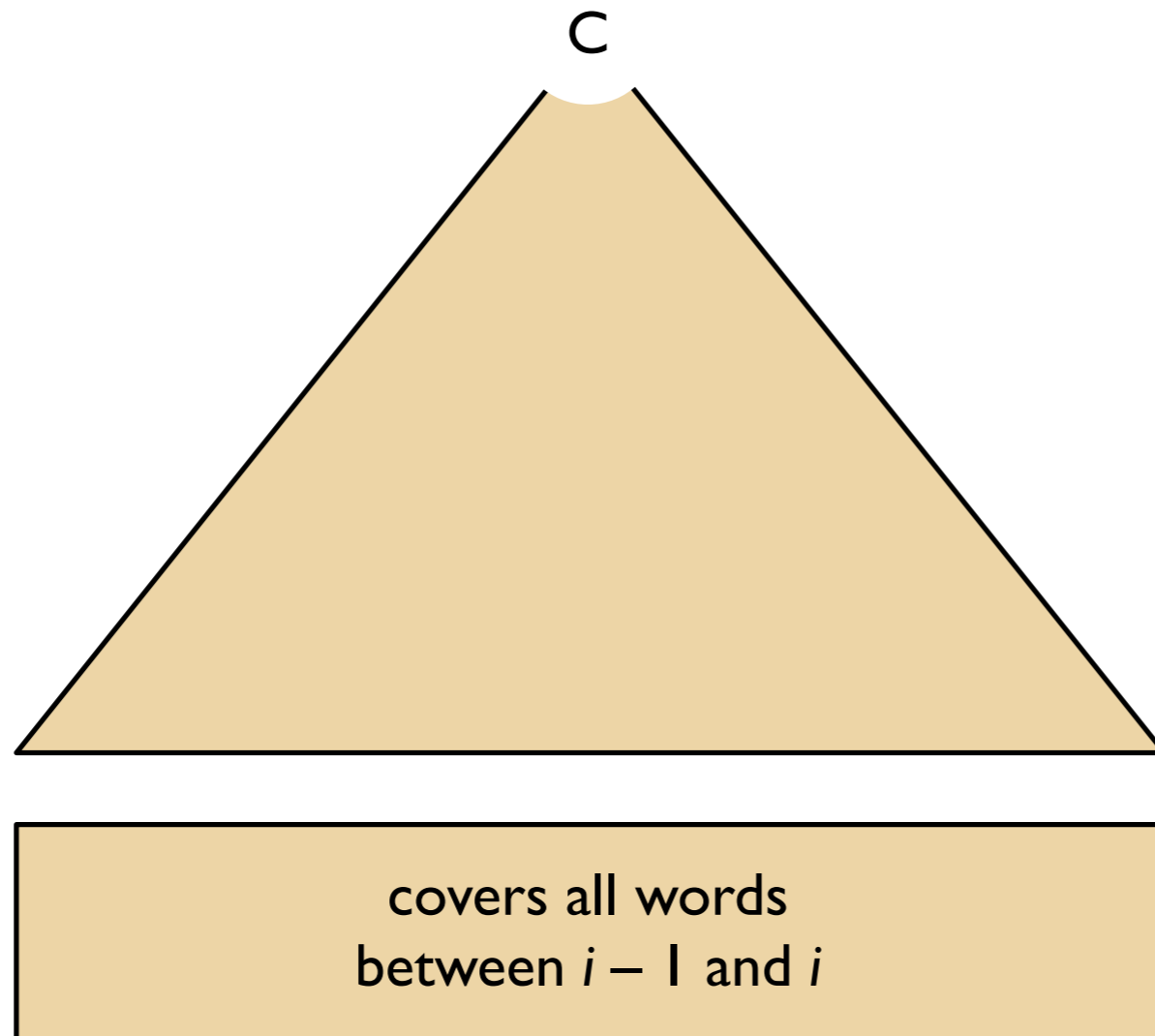




UPPSALA
UNIVERSITET

Recognition

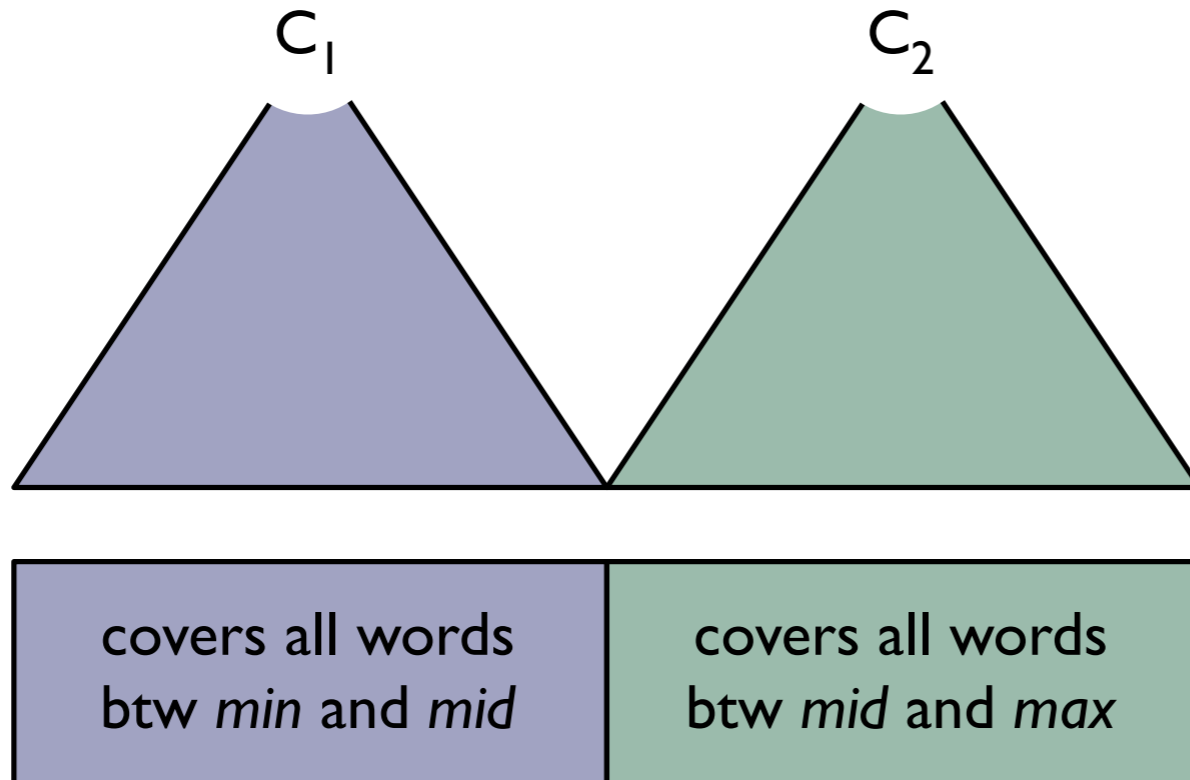
Recognizing small trees





Recognizing big trees

$$C \rightarrow C_1 C_2$$

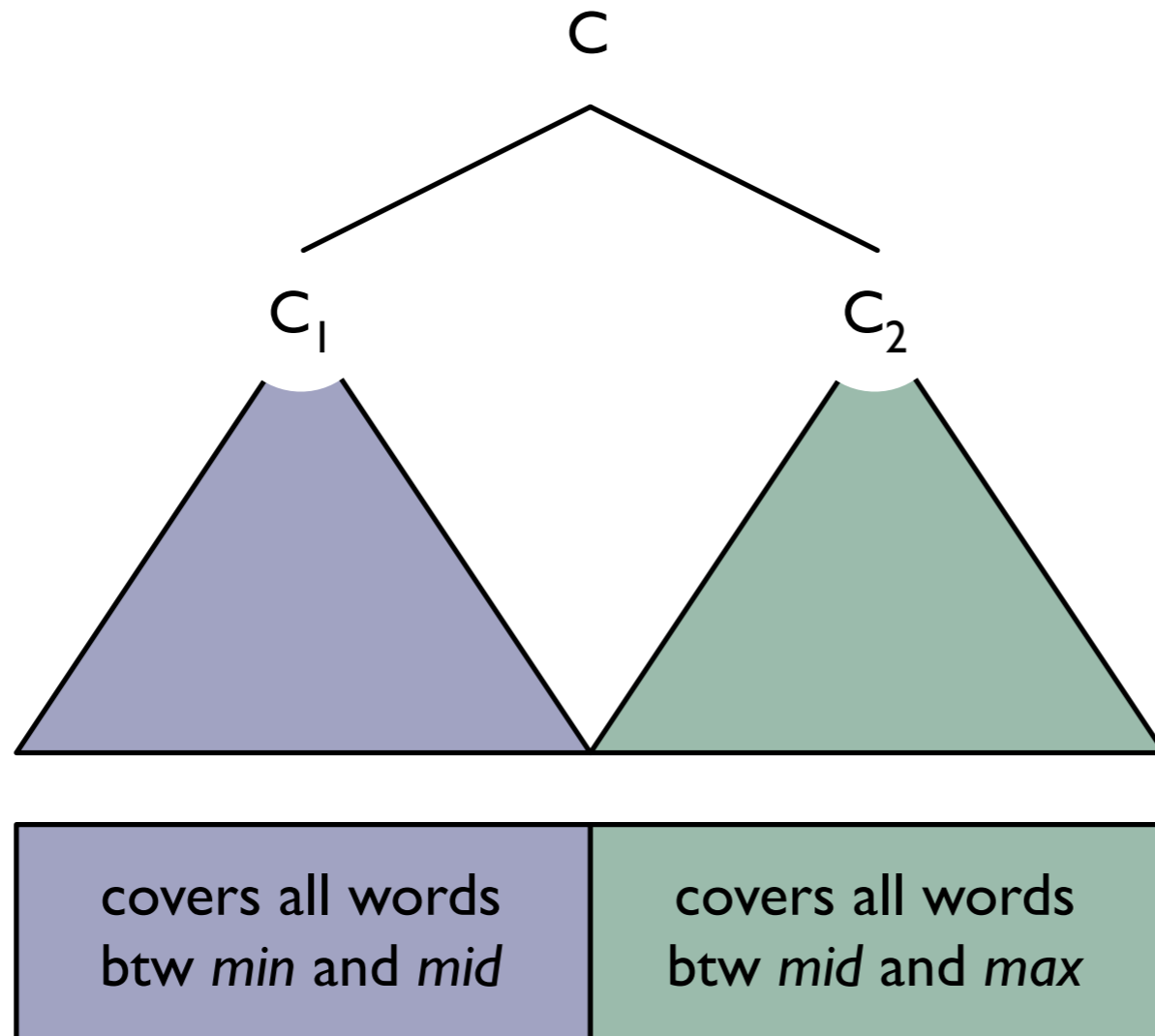




UPPSALA
UNIVERSITET

Recognition

Recognizing big trees

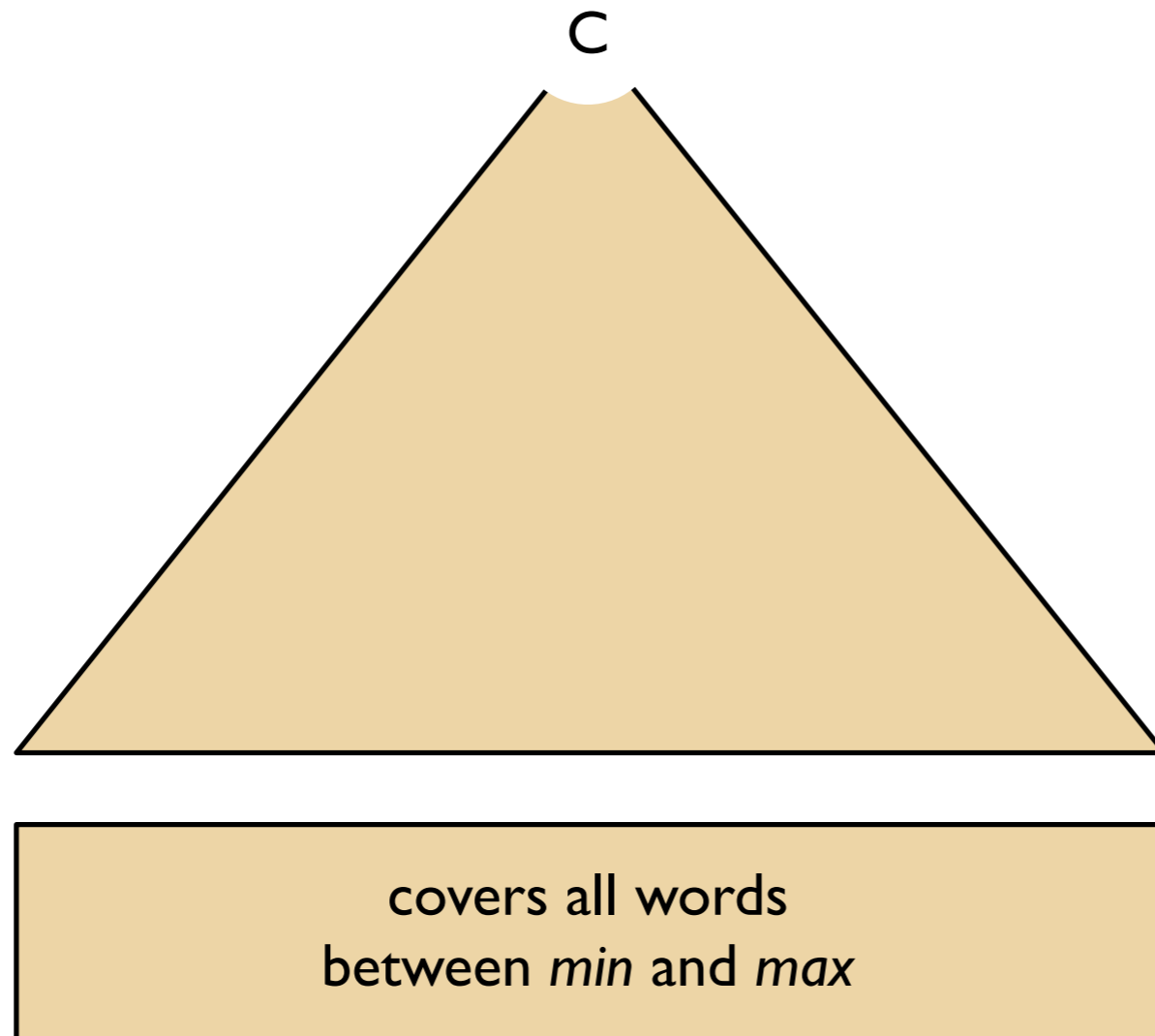




UPPSALA
UNIVERSITET

Recognition

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



UPPSALA
UNIVERSITET

Implementation

Preterminal rules

```
for each  $w_i$  from left to right
```

```
  for each preterminal rule  $C \rightarrow w_i$ 
```

```
    chart[i - 1][i][C] = true
```



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 $C \rightarrow C_1$?
- Why would we want to do that?



Unary rules

```
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  $C \rightarrow C_1$ 
```

```
      if chart[min][max][ $C_1$ ] then
```

```
        chart[min][max][C] = true
```

← new bounds!

Two red arrows originate from the text 'new bounds!' and point to the 'max' and 'min' variables in the code above.



UPPSALA
UNIVERSITET

Implementation

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:**
Apply all rules matching each word
- **For trees built using binary rules:**
Find a binary rule r and a split point mid such that $p(r) \times p(t_1) \times p(t_2)$ is maximal, where t_1 is a most probable left subtree and t_2 is a most probable right subtree.



UPPSALA
UNIVERSITET

Probabilistic parsing

Preterminal rules

for each w_i from left to right

for each preterminal rule $C \rightarrow w_i$

$\text{chart}[i - 1][i][C] = p(C \rightarrow w_i)$



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
```



UPPSALA
UNIVERSITET

Probabilistic parsing

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 $t = \text{chart}[min][max][C]$, 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 w_i from left to right

for each preterminal rule $C \rightarrow w_i$

$\text{chart}[i - 1][i][C] = p(C \rightarrow w_i)$

$\text{backpointerChart}[i-1][i][C] = (C, w_i, i, i-1)$



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
```



Backtrace

Convenient to use recursion to retrieve the tree!

```
# assume backpointers 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

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

- Next week:
 - Sara is traveling; no scheduled teaching
 - Supervision for assignment 1, available by email
- Lab sessions:
 - Monday, Jan 29, 14–15 (after a mini-lecture)
 - Monday, Feb 5, 13–15



Coming up

- Next week: self-studies
 - Assignment I
 - Watch videos + do exercises for coming lectures
- Monday Jan 29 13–15:
 - Mini-lecture + lab
- Next theme:
 - Transition-based dependency parsing
 - Recorded lectures and exercises are available in Studium
 - Lecture Wednesday Jan 31, **8–10 (NOTE!)**
- Seminar I: Wednesday February 7, 9–12
 - Groups+times will be posted on the web page



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!**
 - **Don't be afraid to contact me for help!**



Assignment 1: small bug

- Small issue with the debugging code for CNF
- Line 42:
 - `print("Output: " + str(dumps(tree)), file=stderr)`
 - Change it to:
 - `print("Output: " + str(dumps(conv_tree)), file=stderr)`
- Updated in the code to download