UPPSALA UNIVERSITET

The CKY algorithm part 2: Probabilistic parsing

Syntactic analysis (5LN455)

2013-11-20

Sara Stymne
Department of Linguistics and Philology

Based on slides from Marco Kuhlmann

UPPSALA UNIVERSITET

Recap: The CKY algorithm

UPPSALA UNIVERSITET

The CKY algorithm

The CKY algorithm is an efficient bottom–up parsing algorithm for context-free grammars.

We use it to solve the following tasks:

- Recognition:
  Is there any parse tree at all?
- Probabilistic parsing:
  What is the most probable parse tree?

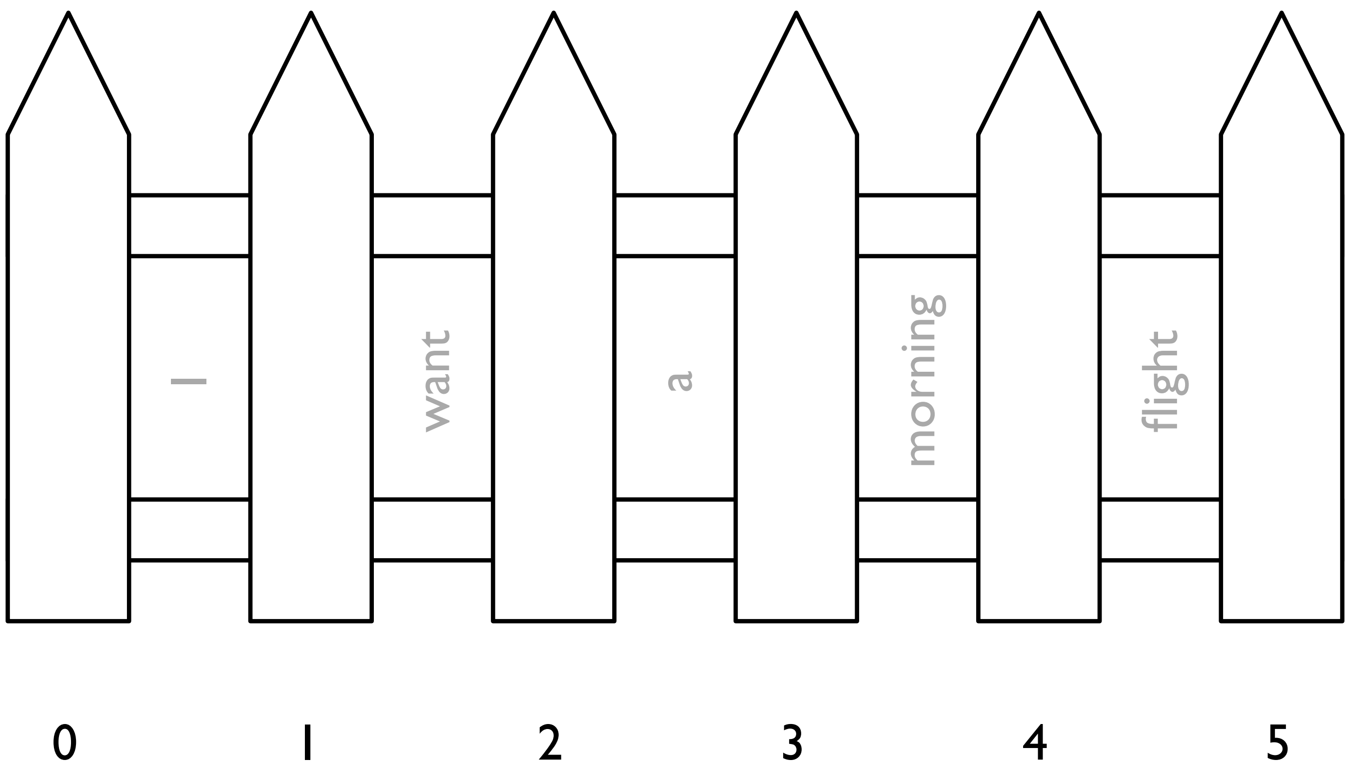
Restrictions

- The CKY algorithm as we present it here can only handle rules that are at most binary:
$C \rightarrow w_i$, $C \rightarrow C_1$, $C \rightarrow C_1 \ C_2$.
- 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 (J&M 13.4.2).

UPPSALA UNIVERSITET

Fencepost positions

We view the sequence w as a fence with n holes, one hole for each token w_i, and we number the fenceposts from 0 till n.



UPPSALA UNIVERSITET

Implementation

- The implementation uses a three-dimensional array *chart*.
- Whenever we have recognized a parse tree that spans all words between *min* and *max* and whose root node is labeled with *C*, we set the entry *chart*[*min*][*max*][*C*] to *true*.

UPPSALA UNIVERSITET

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
```

UPPSALA UNIVERSITET

Question

How do we need to extend the code in order to handle unary rules $C \rightarrow C_l$?

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
```

Question

This is not quite right.

Why, and how could we fix the problem?

UPPSALA UNIVERSITET

Structure

- Is there any parse tree at all?
- What is the most probable parse tree?

UPPSALA UNIVERSITET

Probabilistic parsing

What is the most probable parse tree?

- The number of possible parse trees grows rapidly with the length of the input.
- But not all parse trees are equally useful.

  Example: I booked a flight from Los Angeles.
- In many applications, we want the 'best' parse tree, or the first few best trees.
- Special case: 'best' = 'most probable'

Probabilistic context-free grammars

A probabilistic context-free grammar (PCFG) is a context-free grammar where

- each rule r has been assigned a probability $p(r)$ between 0 and 1
- the probabilities of rules with the same left-hand side sum up to 1

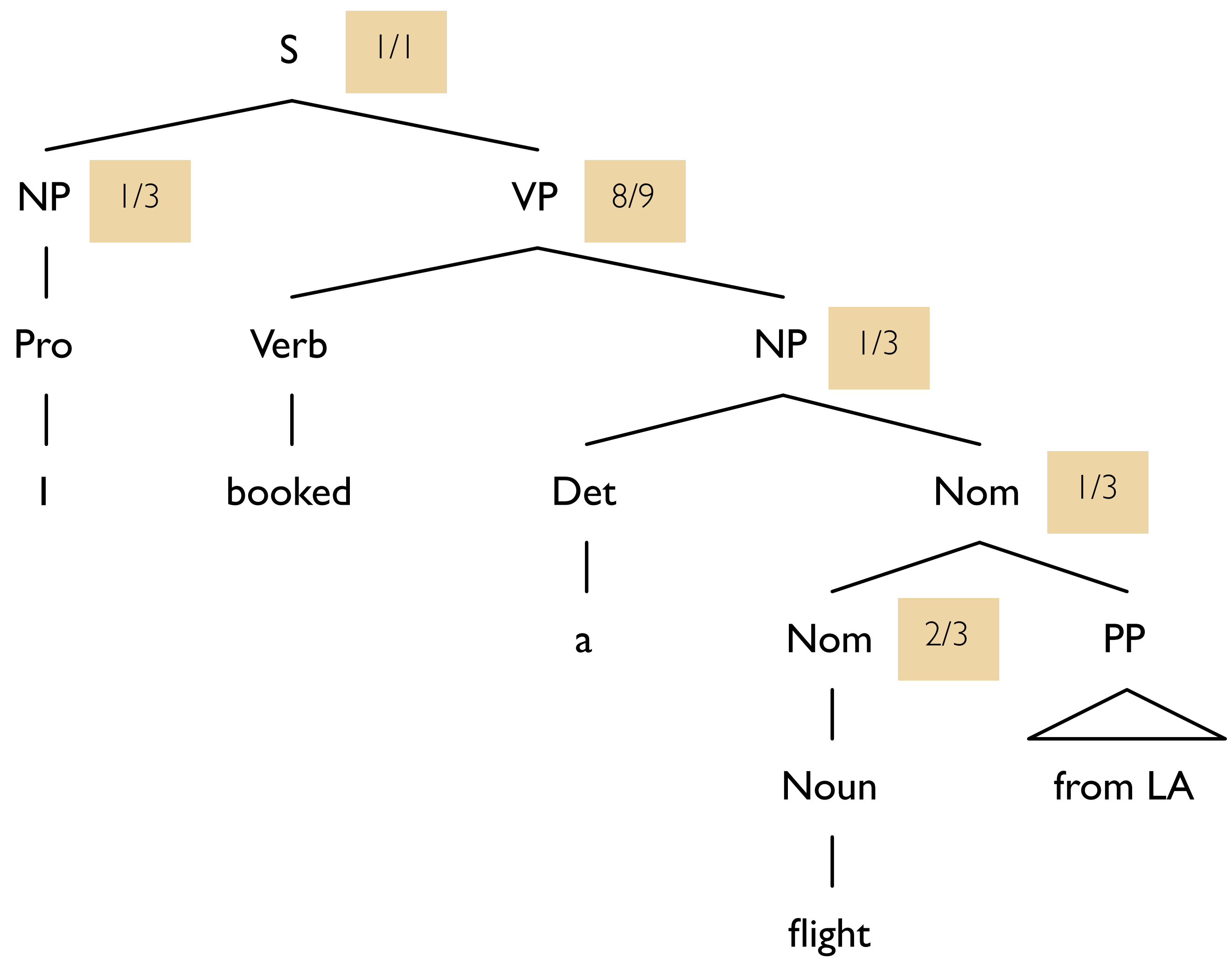
Example

Rule	Probability
S → NP VP	1
NP → Pronoun	1/3
NP → Proper-Noun	1/3
NP → Det Nominal	1/3
Nominal → Nominal PP	1/3
Nominal → Noun	2/3
VP → Verb NP	8/9
VP → Verb NP PP	1/9
PP → Preposition NP	1

The probability of a parse tree

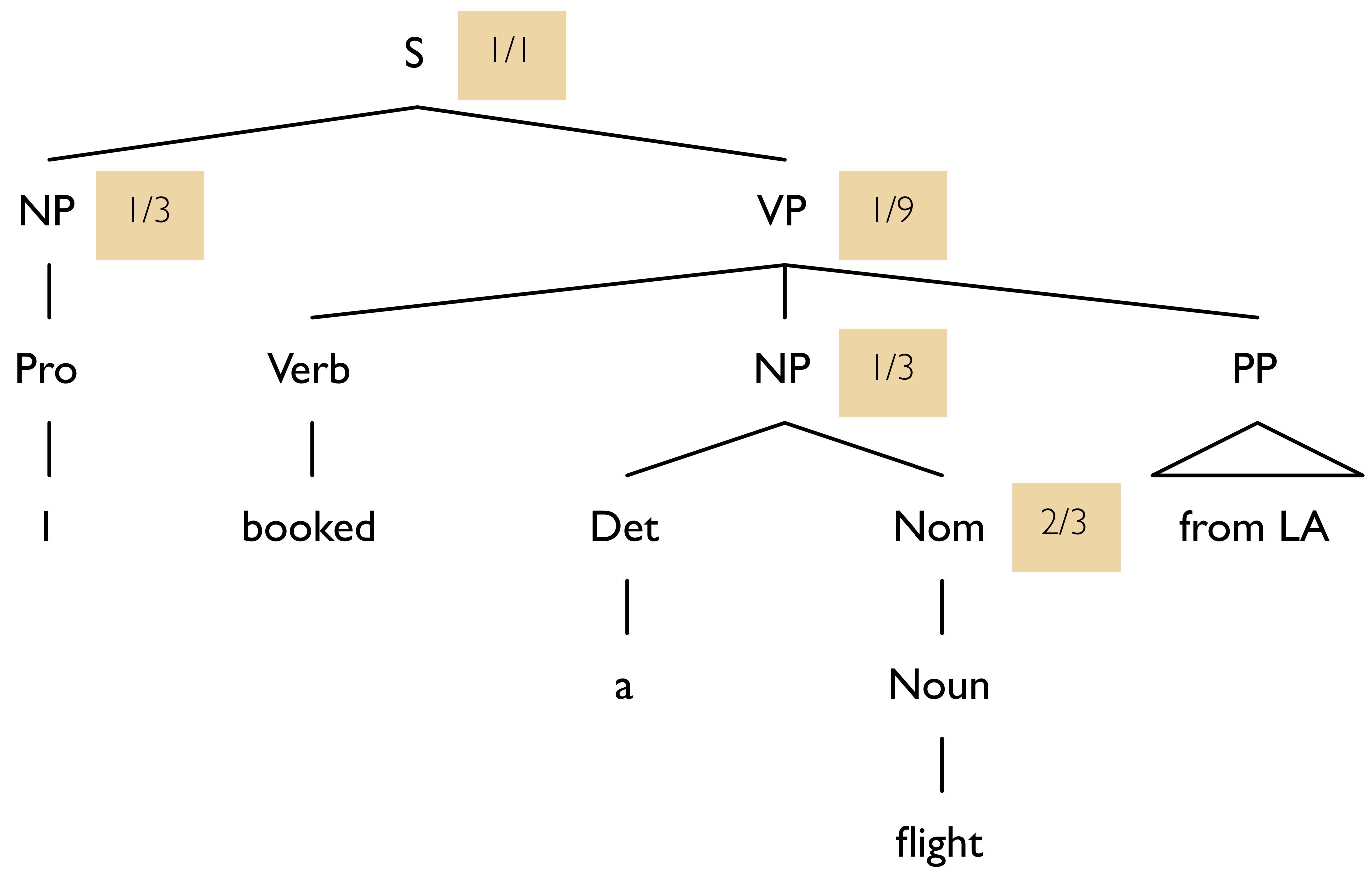
The probability of a parse tree is defined as the product of the probabilities of the rules that have been used to build the parse tree.

Example



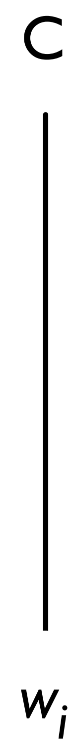
Probability: 16/729

Example



Probability: 6/729

Small trees



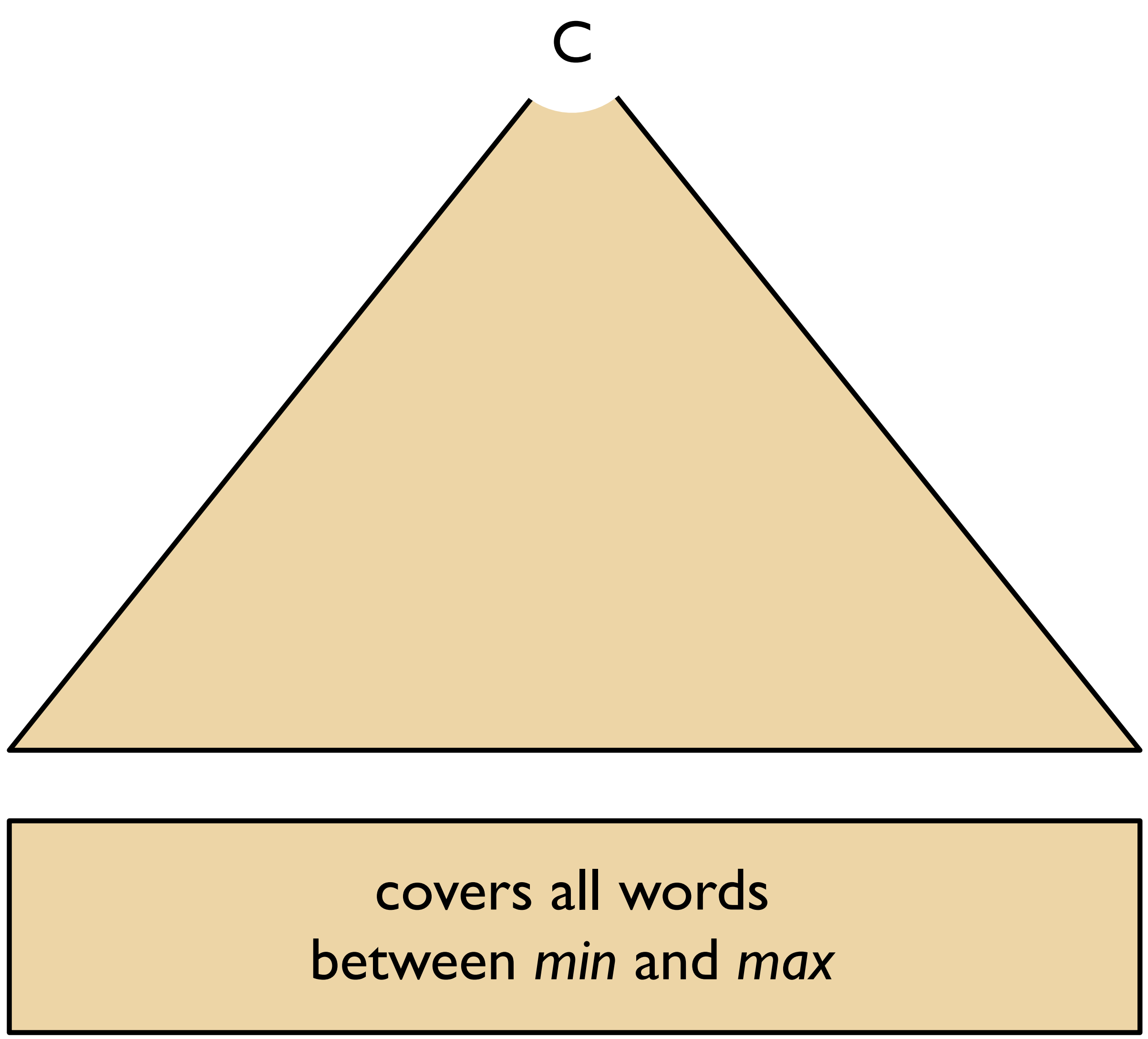
Small trees

$C \rightarrow w_i$

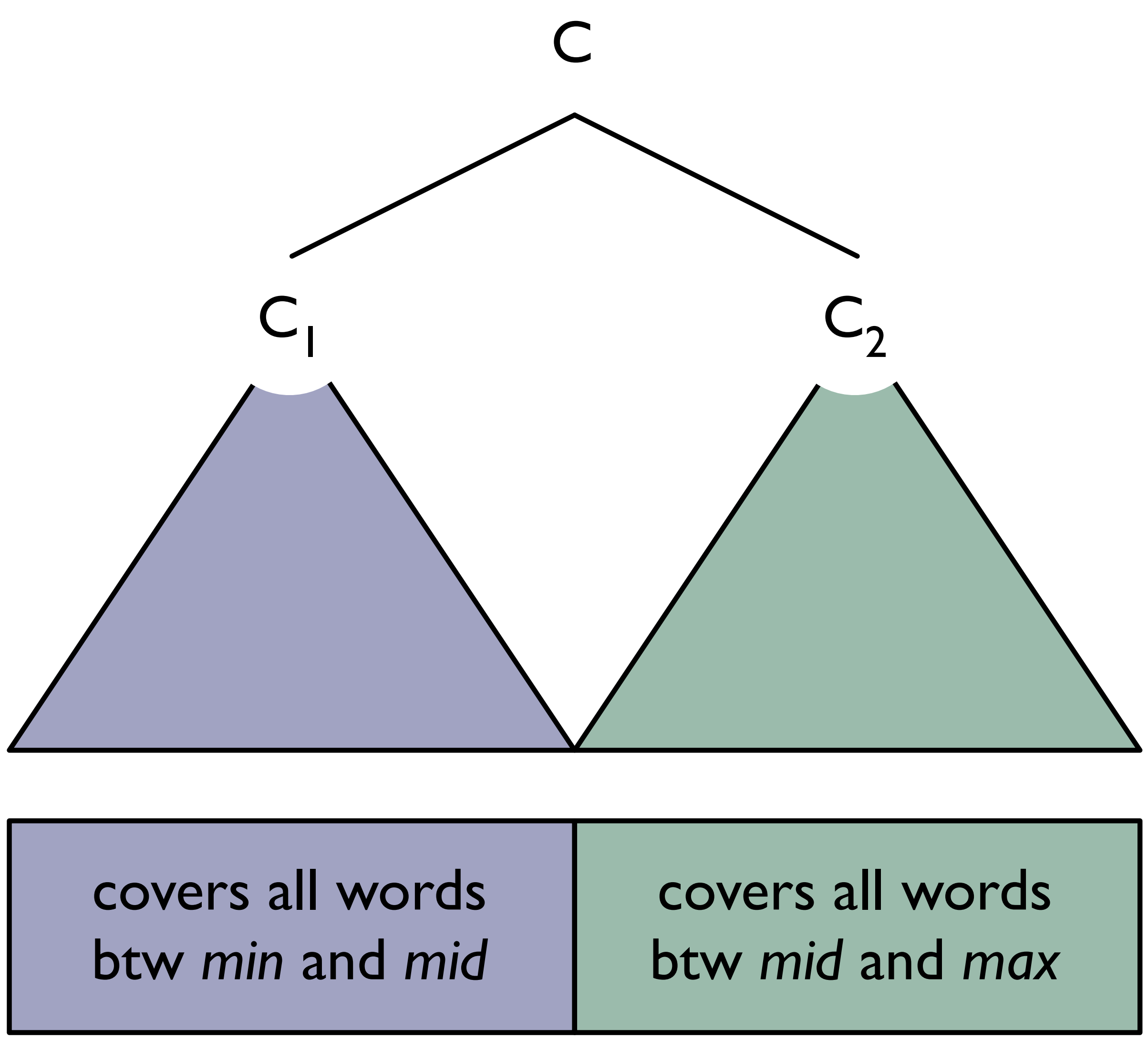
Choose the most probable rule!

w_i

Big trees



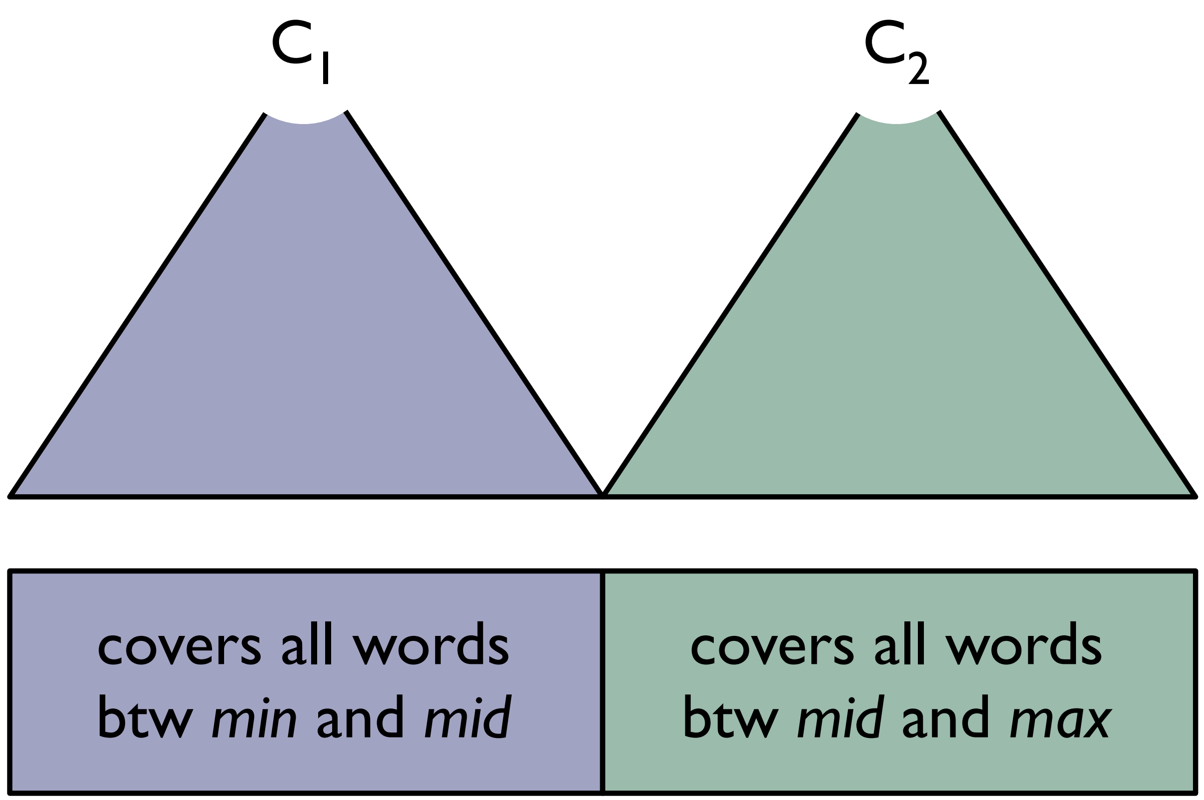
Big trees


C
C1
C2
covers all words btw *min* and *mid*
covers all words btw *mid* and *max*

Big trees

$C \rightarrow C_1\ C_2$

Choose the most probable rule!



Idea

- For trees built using preterminal rules:
  Find a most probable rule.

- 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.

Implementation

- Instead of an array with Boolean values, we now have an array with probabilities, i.e., *doubles*.
- When all is done, we want to have
  chart[*min*][*max*][*C*] = *p*
  if and only if a most probable parse tree with signature [*min*, *max*, *C*] has probability *p*.

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?

One more question

The only thing that we have done so far is to compute the *probability* of the most probable parse tree. But how does that parse tree look like?

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.

Backpointers

```
double best = undefined

Backpointer backpointer = undefined

...

if candidate > best then

  best = candidate

  // We found a better tree; update the backpointer!

  Backpointer bp1 = backpointerChart[min][mid][C1]

  Backpointer bp2 = backpointerChart[mid][max][C2]

  backpointer = new Backpointer(C -> C1 C2, bp1, bp2)

...

chart[min][max][C] = best

backpointerChart[min][max][C] = backpointer
```

Skeleton code

```
// int n = number of words in the sequence

// int m = number of syntactic categories in the grammar

// int s = the (number of the) grammar's start symbol

double[][][] chart = new double[n + 1][n + 1][m]

Backpointer[][][] bpChart = new BackPointer[n + 1][n + 1][m]

// Recognize all parse trees built with with preterminal rules.

// Recognize all parse trees built with inner rules.

if chart[0][n][s] > 0 then  //or bpChart[0][n][s] != null

    return build_tree(bpChart[0][n][s]);

return null;
```

UPPSALA UNIVERSITET

Summary

- The CKY algorithm is an efficient parsing algorithm for context-free grammars.
- Today, we have used it for probabilistic parsing: The task of computing the most probable parse tree for a given sentence.
- Reading: J&M sections 14.1, 14.2