UPPSALA
UNIVERSITET

The CKY algorithm part 1: Recognition

Syntactic parsing

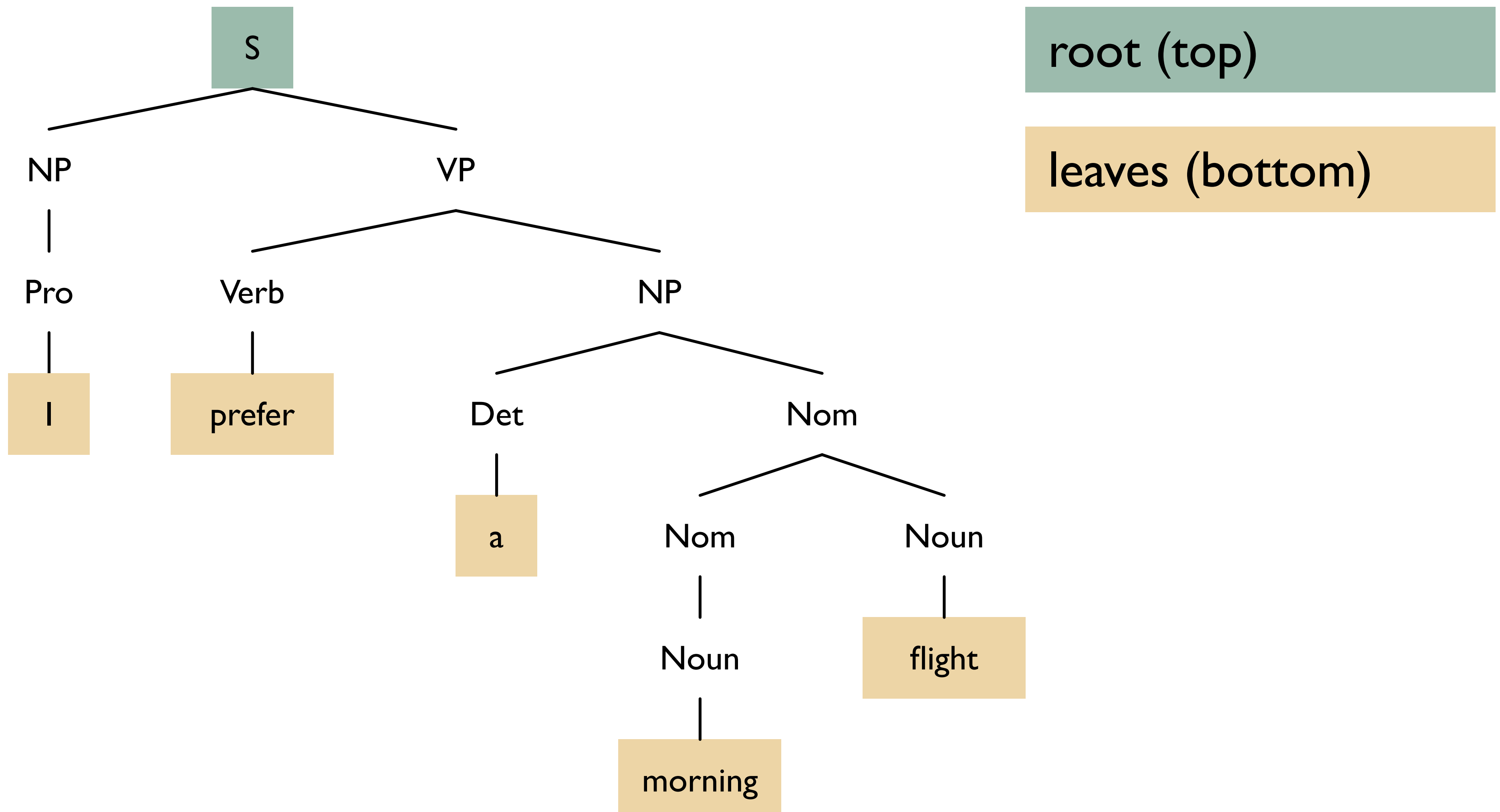
2018-01-17

Sara Stymne
Department of Linguistics and Philology

Mostly based on slides from Marco Kuhlmann

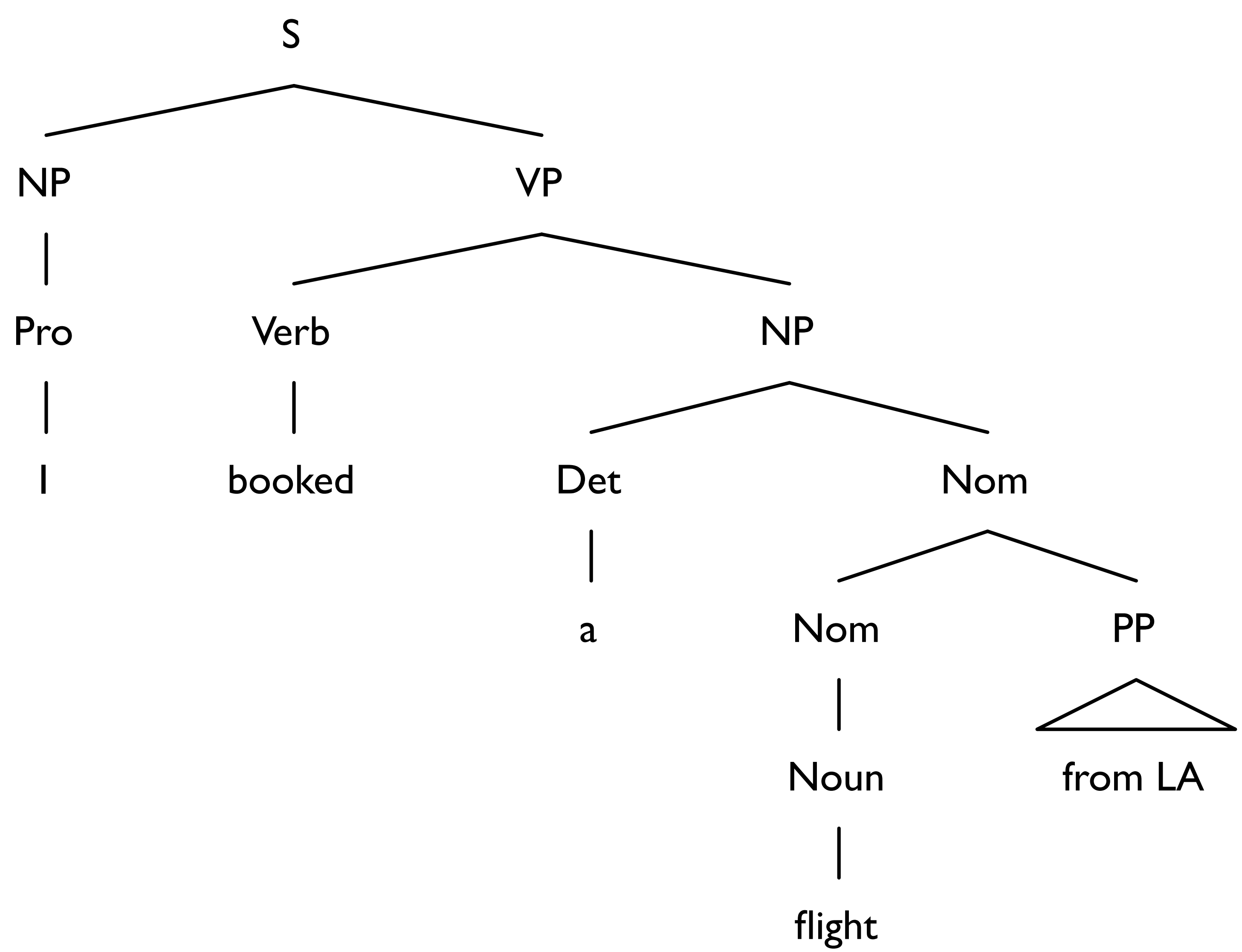
UPPSALA UNIVERSITET

Phrase structure trees



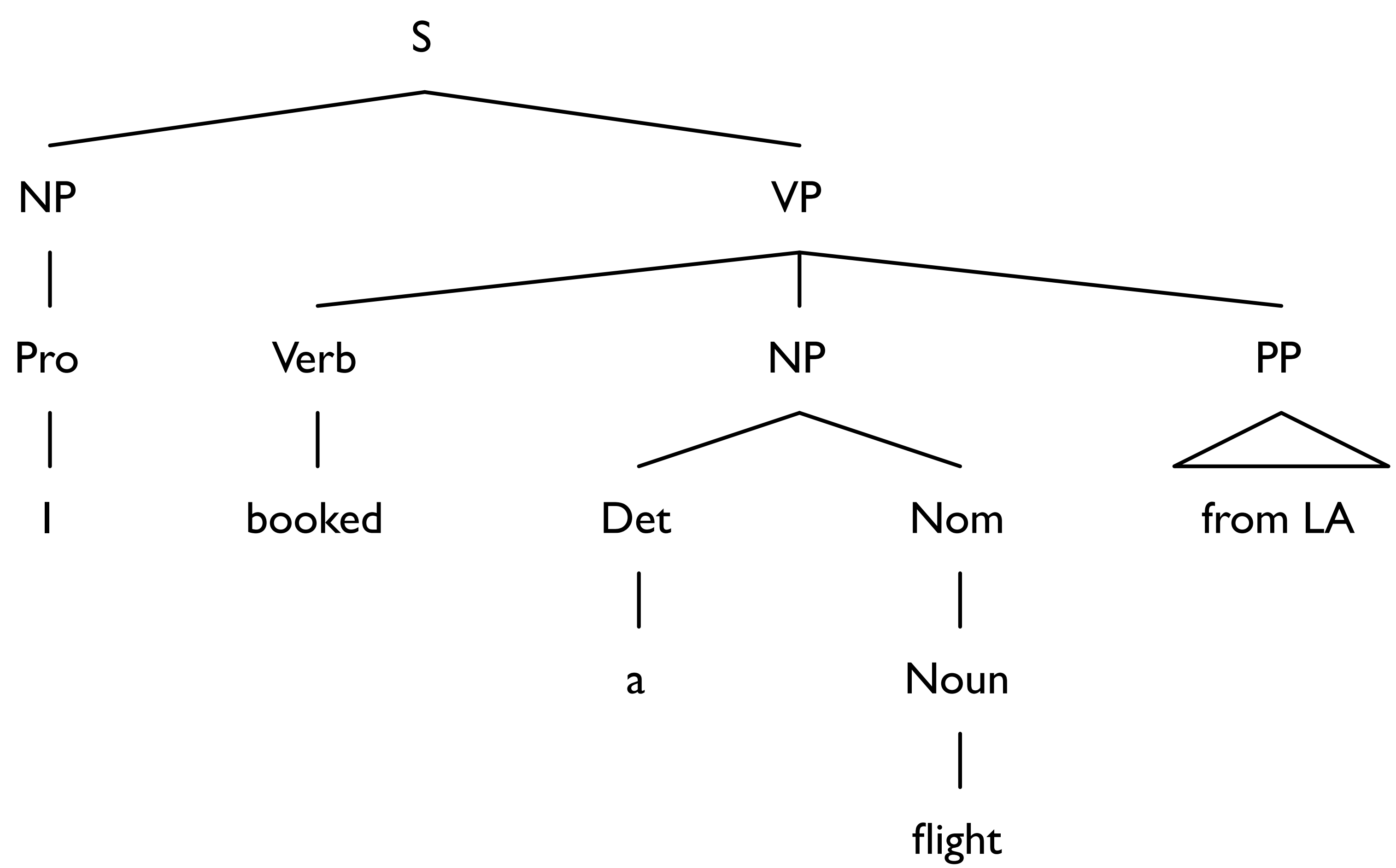
UPPSALA UNIVERSITET

Ambiguity



UPPSALA UNIVERSITET

Ambiguity



UPPSALA UNIVERSITET

Parsing as search

- Parsing as search:
  search through all possible parse trees
  for a given sentence
- bottom–up:
  build parse trees starting at the leaves
- top–down:
  build parse trees starting at the root node

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.

Applications

The CKY algorithm can be used to compute many interesting things.
Here 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 original CKY algorithm can only handle rules that are at most binary:
  $C \to w_i\ ,\ C \to C_1\ C_2\ .$
- It can easily be extended to also handle unit productions:
  $C \to w_i\ ,\quad C \to C_1\ ,\quad C \to C_1\ C_2\ .$
- 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

Restrictions - details

- The CKY algorithm originally handles grammars in CNF (Chomsky normal form):
  $C \rightarrow w_i$, $C \rightarrow C_1 \; C_2$, $(S \rightarrow \varepsilon)$
- ε is normally not used in natural language grammars
- This is what you will use in assignment 2
- We will also discuss allowing unit productions, $C \rightarrow C_1$
  - Extended CNF
  - Easy to integrate into CKY, gives easier grammar conversions

UPPSALA UNIVERSITET

Conversion to CNF

- Eliminate mixed rules:
  - VP->V to VP -- VP->V INF VP, INF->to
- Elimainate n-ary branching subtrees, with n>2, by inserting additional nodes
  - VP->V INF VP -- VP->V X1, X1->INF V
- Eliminate unary branching by merging nodes
  - S-> NP VP, NP->PRON, PRON->you -- NP->you

Conversion to CNF

- Eliminate mixed rules:
  - VP->V to VP -- VP->V INF VP, INF->to
- Eliminate n-ary branching subtrees, with n>2, by inserting additional nodes
  - VP->V INF VP -- VP->V X1, X1->INF V

    more readable: VP->V VP|V, VP|V->INF VP
- Eliminate unary branching by merging nodes
  - S-> NP VP, NP->PRON, PRON->you -- NP->you

    more readable: NP->NP+PRON VP, NP+PRON->you

UPPSALA UNIVERSITET

Conversion to CNF

- The preceding slide showed how to convert a grammar to CNF
- It is also possible to convert a treebank to CNF
  - You will do this in task 1

UPPSALA UNIVERSITET

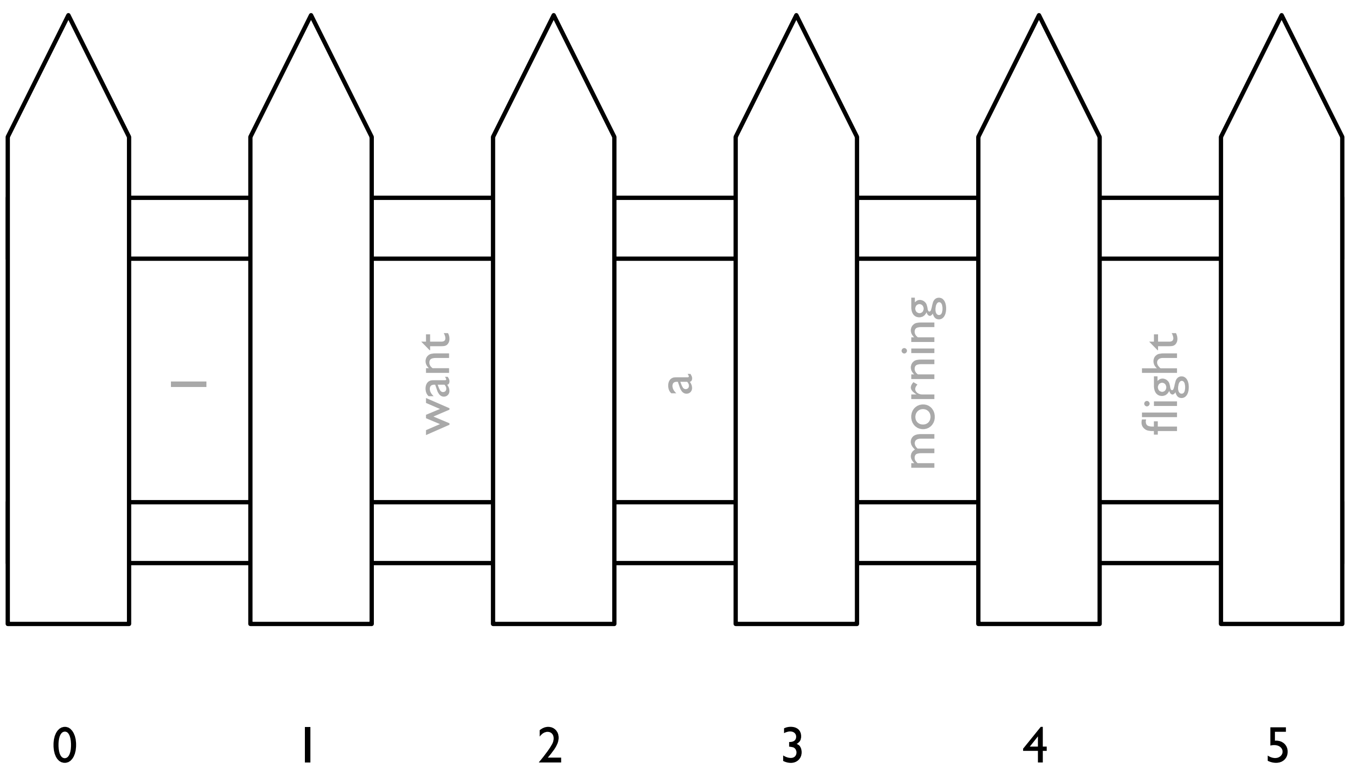
Conventions

- We are given a context-free grammar G and a sequence of word tokens $w = w_1 \ldots w_n$.
- We want to compute parse trees of w according to the rules of G.
- We write S for the start symbol of G.

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

Structure

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

Recognition

Recognizer

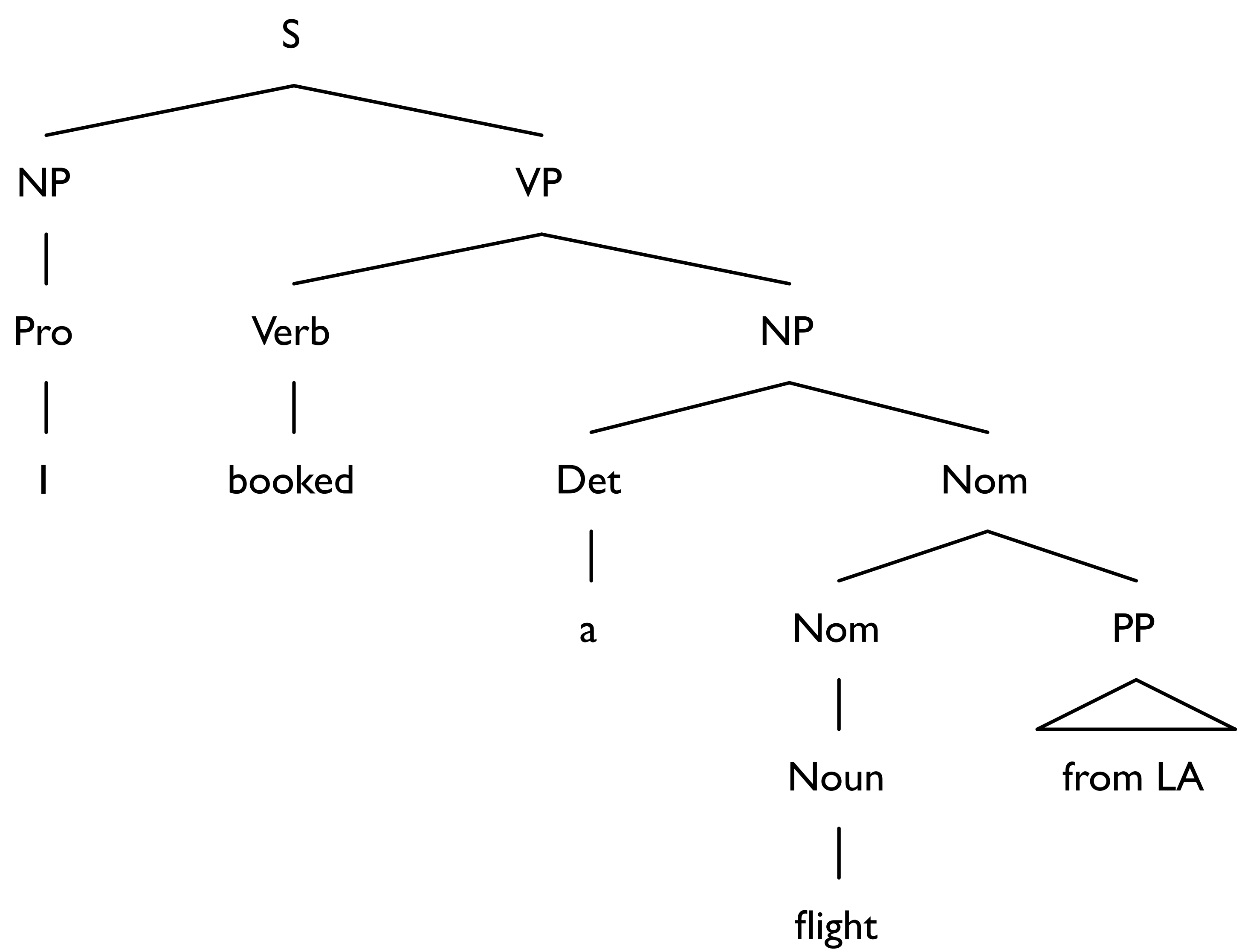
A computer program that can answer the question

> Is there any parse tree at all
> for the sequence *w* according to the grammar *G*?

is called a recognizer.

In practical applications one also wants
a concrete parse tree, not only an answer
to the question whether such a parse tree exists.

Parse trees



Preterminal rules and inner rules

- preterminal rules:
  rules that rewrite a part-of-speech tag to a token, i.e. rules of the form $C \rightarrow w_i$

  Pro → I, Verb → booked, Noun → flight

- inner rules:
  rules that rewrite a syntactic category to other categories: $C \rightarrow C_1\ C_2$, $(C \rightarrow C_1)$

  S → NP VP, NP → Det Nom, (NP → Pro)

Recognizing small trees

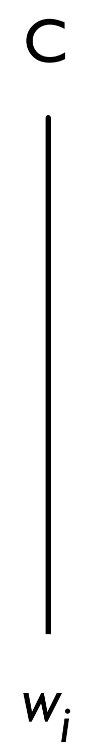
w_i

Recognizing small trees

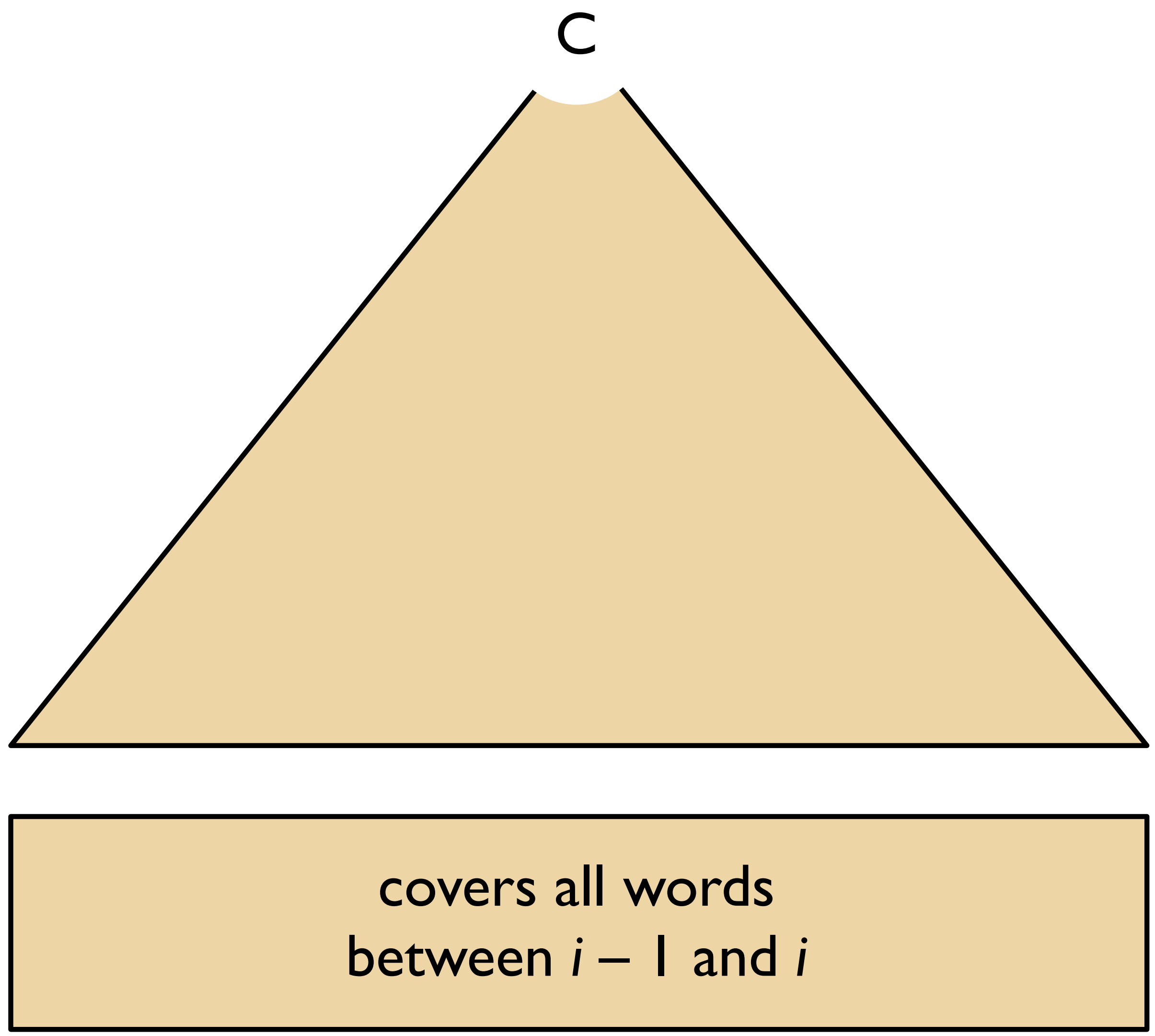
$C \rightarrow w_i$

w_i

Recognizing small trees

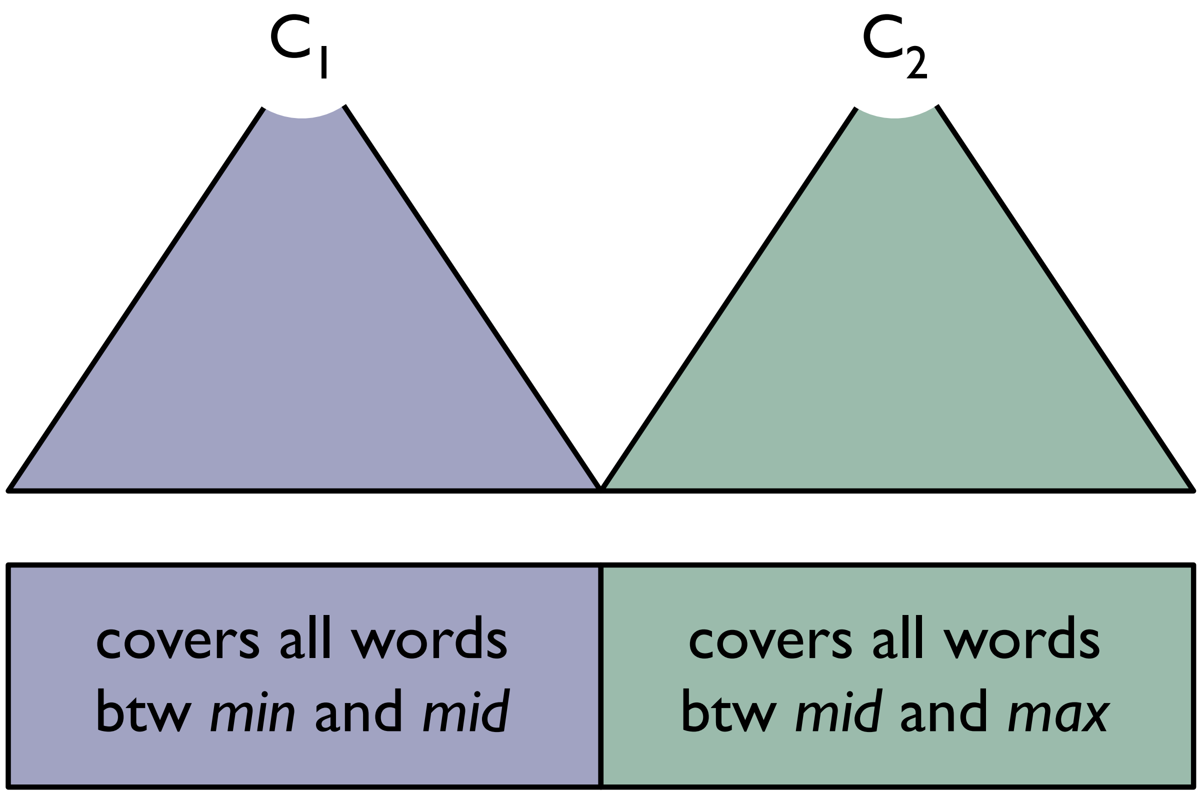


Recognizing small trees



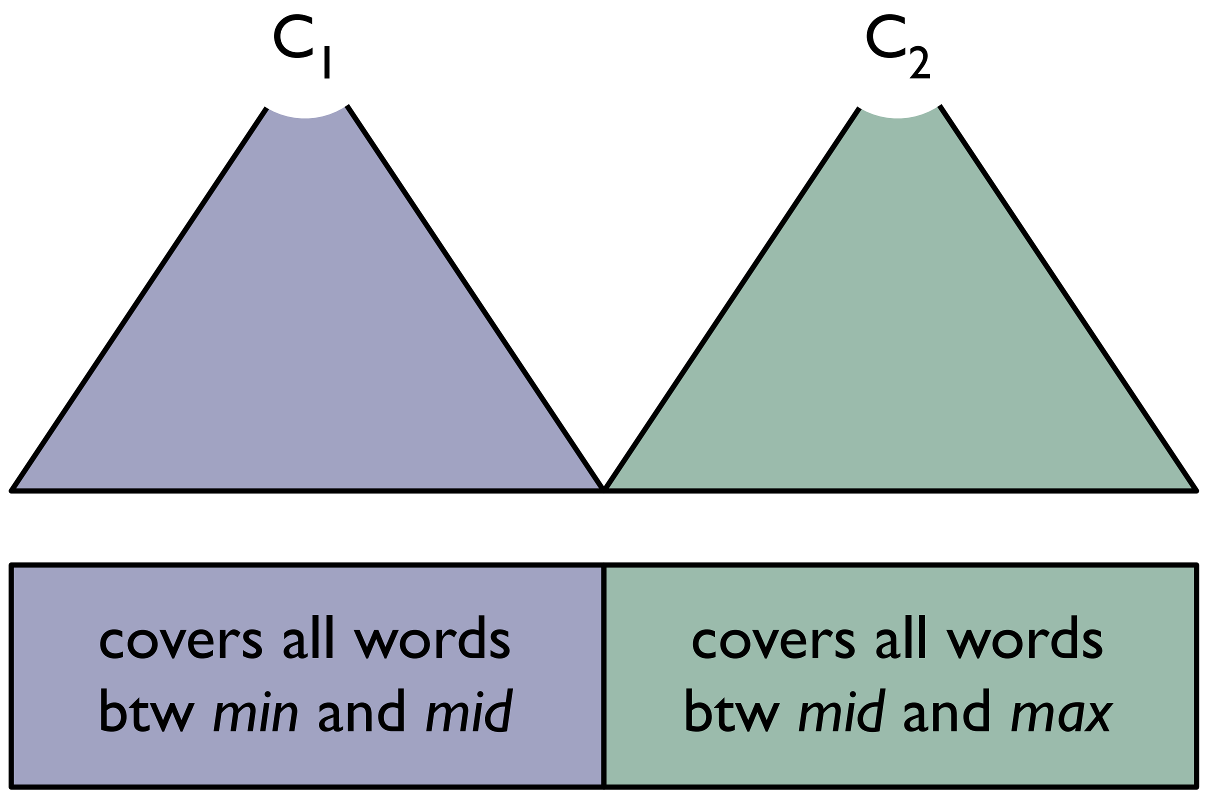
UPPSALA UNIVERSITET

Recognizing big trees

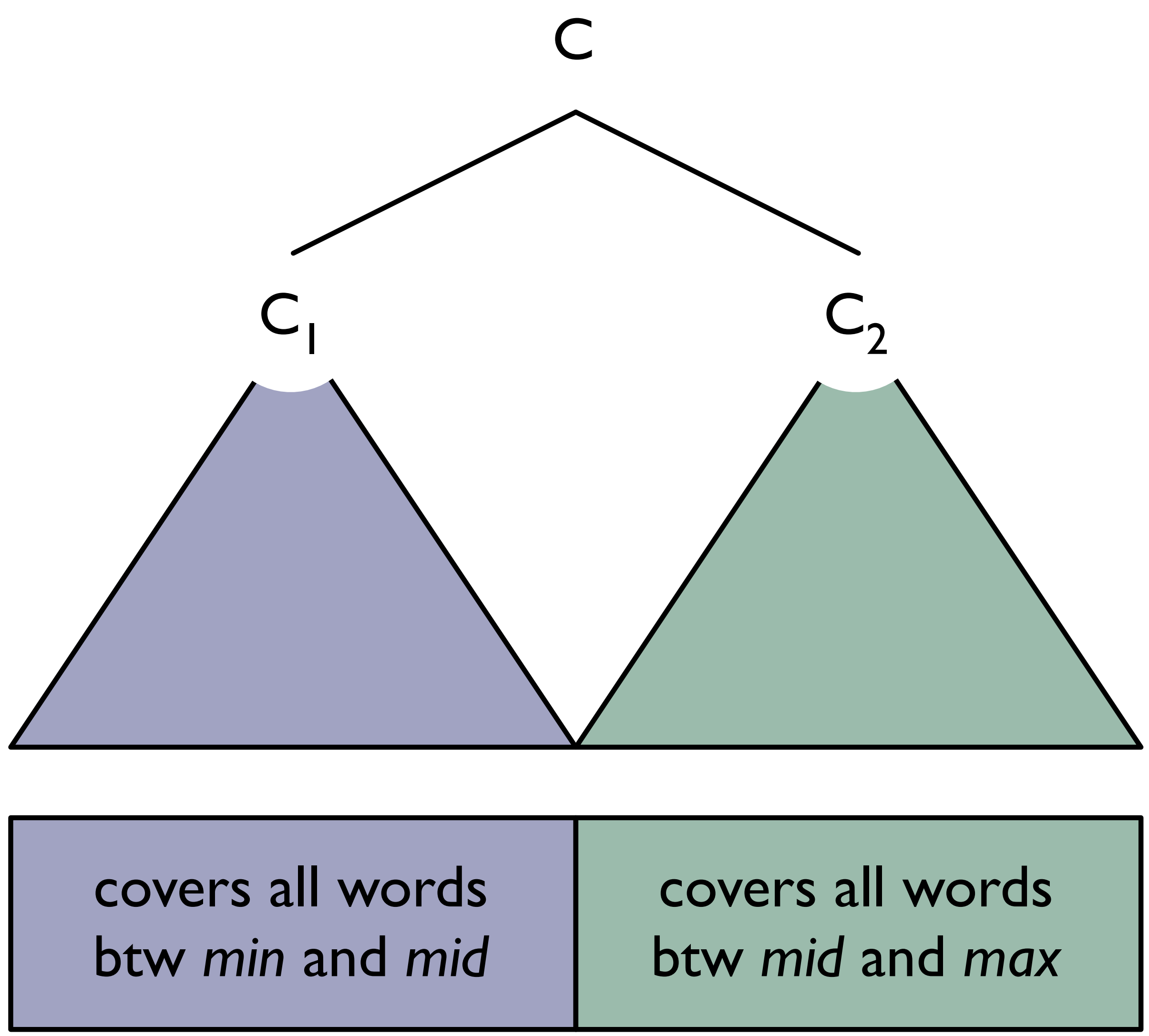


Recognizing big trees

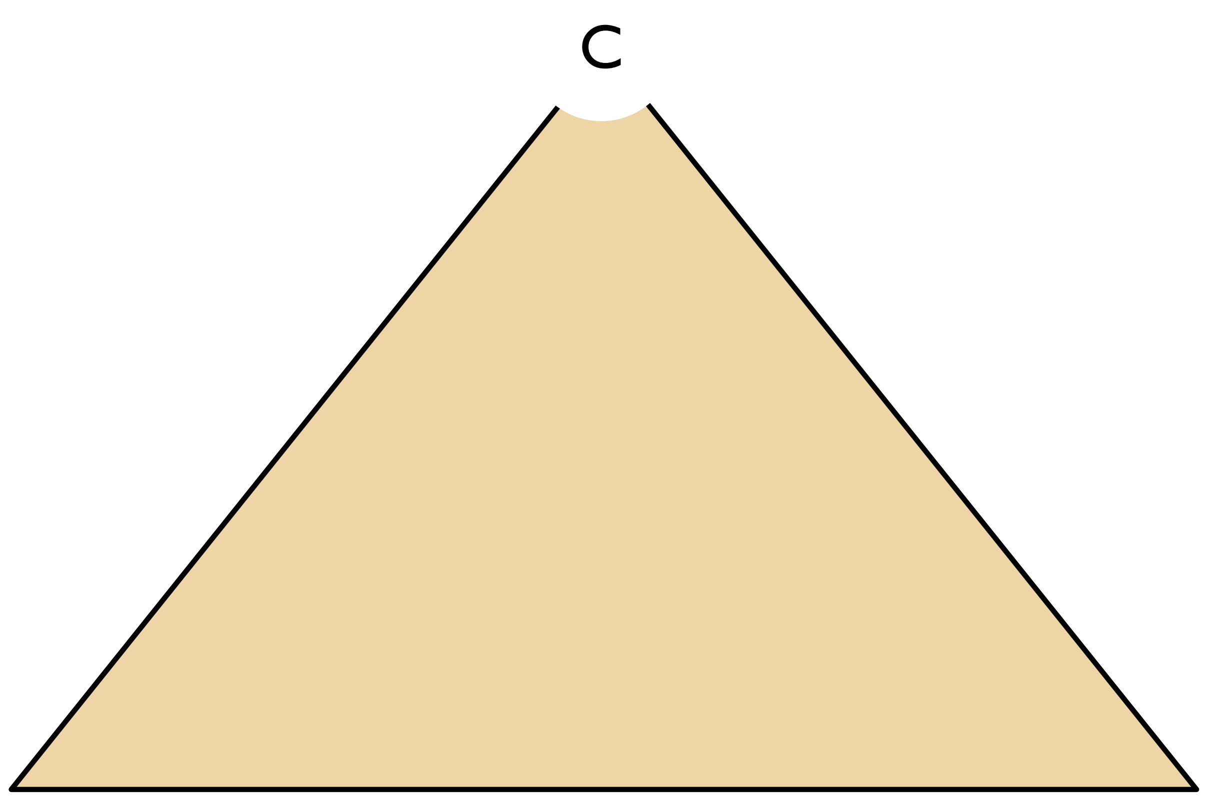
$C \rightarrow C_1\ C_2$



Recognizing big trees



Recognizing big trees



covers all words
between *min* and *max*

Questions

- 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: $C \rightarrow C_l$?

Signatures

- The rules that we have just seen are independent of a parse tree's inner structure.
- The only thing that is important is how the parse tree looks from the 'outside'.
- We call this the signature of the parse tree.
- A parse tree with signature [*min*, *max*, *C*] is one that covers all words between *min* and *max* and whose root node is labeled with *C*.

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?

Implementation

Data structure

- The standard implementation represents signatures by means of a three-dimensional array *chart*.
- Initially, all entries of *chart* should be set to *false*.
- 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*.

Pseudo code

- Informal high-level description, of how a computer program or algorithm works
- Meant to be read and understood by humans, not machines
- Can be augmented:
  - Natural language descriptions
  - Compact mathemtical notation
- Efficient description of key principles of an algorithm, indeendently of programming languages and environments
- Will be used to describe parsing algorithms on slides, and in books
  - Your assingment task 1 is to ”translate” pseudo code to python

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
```

Numbering of categories

- In order to use standard arrays, we need to represent syntactic categories by numbers.
- We write m for the number of categories; we number them from 0 till $m - 1$.
- We choose our numbers such that the start symbol S gets the number 0.

CKY in python

- A three-dimensional array might not be the most suitable choice in python (even though it'd work).
- It is quite possible to use more python-lika data structures like dictionaries, or variants such as defaultdict
  - Use tuples as keys, e.g. `(i,j,S)`; ex: `(2,3,"Pron")`
  - Lookup in chart: `chart[i,j,S]`
  - No need to numberize categories in this solution

Questions

- 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

Summary

- The CKY algorithm is an efficient parsing algorithm for context-free grammars.
- Today: Recognizing whether there is any parse tree at all.
- Next time: Probabilistic parsing – computing the most probable parse tree.

UPPSALA UNIVERSITET

Reading

- Recap of the introductory lecture:
  J&M chapter 12.1-12.7 and 13.1-13.3
- CKY recognition:
  J&M section 13.4.1
- CKY probabilistic parsing, for next week:
  J&M section 14.1-14.2