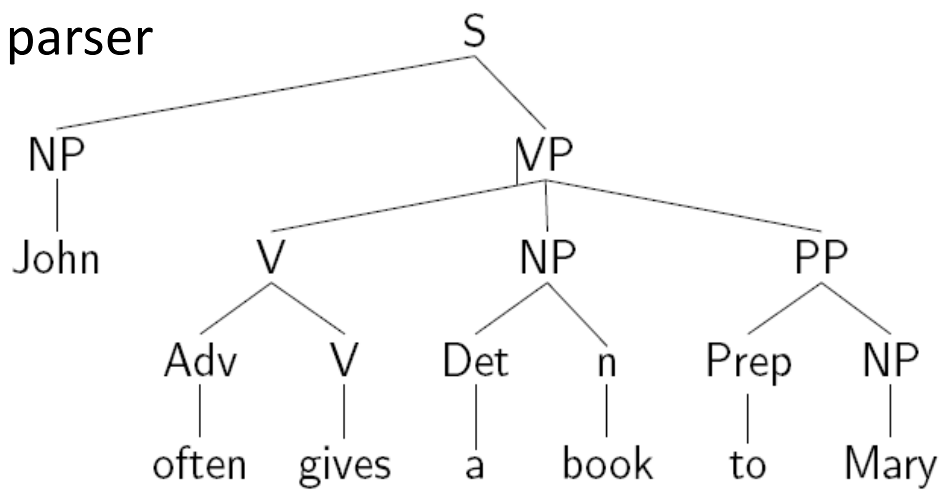
- Jurafsky, D. and Martin, J. H. (2009): Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Second Edition. Pearson: New Jersey: Chapter 13

Chunking, Syntax trees, Context-Free Grammar parsing

SYNTAX PROCESSING

SYNTAX, GRAMMARS, PARSING

- Syntax captures structural relationships between words and phrases, i.e. describes the constituent structure of NL expressions
- Constituents: Noun Phrase, Verb Phrase, Determiners....
- Grammars are used to describe the syntax of a language, cf. Context Free Grammars in Lecture 1
- Syntactic analyzers assign a syntactic structure to a string on the basis of a grammar
- A syntactic analyzer is also called a parser



MOTIVATION FOR SYNTAX PROCESSING

- Natural language is more than trigrams
- For ‘understanding’ language better, we want to be able to recognize syntactic structures
- These are again just another layer of processing
- For now: we ignore meaning and simply look at syntax.
  I.e. “colorless green ideas sleep furiously” is syntactically correct

What level of syntactic processing is ‘right’? Depends on the goal.

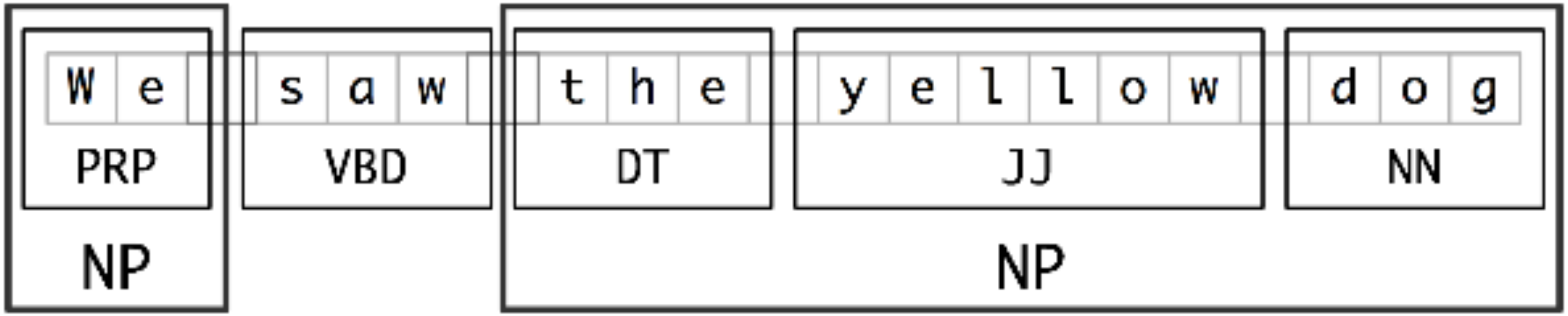
- Chunking / partial parsing: only continuous chunks
- dependency parsing
- phrase structure grammars
- constituency
- attribute value grammars

CHUNKING (PARTIAL PARSING)

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- `[I begin] [with an intuition]: [when I read] [a sentence], [I read it] [a chunk] [at a time]`
  (Example from S. Abney, Parsing by Chunks)
- chunks correspond to prosodic patterns – where to put the breaks when reading the sentence aloud
- chunks are typically subsequences of grammatical constituents: noun groups and verb groups
- chunks are non-overlapping, non-nested regions of text
- chunking is a kind of higher level label segmentation
- Usually, chunks contain a head, with the possible addition of modifiers and function words
  [quickly **walk**] [straight **past**] [the **lake**]
- Most interesting for applicatons: NP chunks

CHUNKING VIEWED AS SEGMENTATION



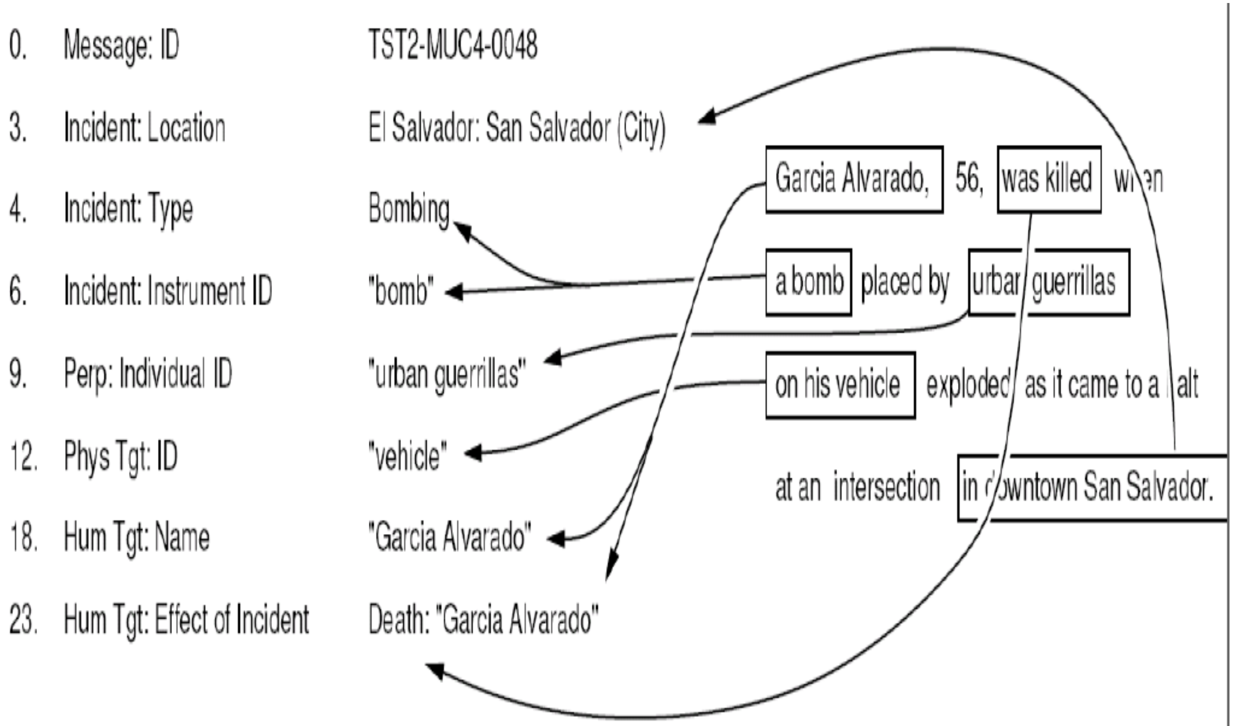
Segmentation and labeling of multi-token sequences

- Smaller boxes: word-level segmentation and labeling
- Large boxes: higher-level segmentation and labeling

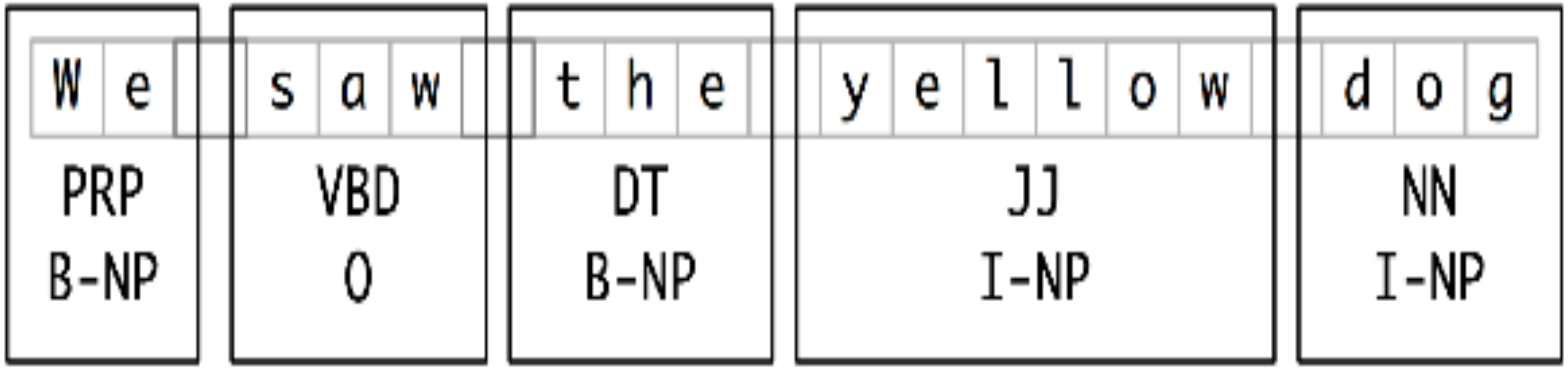
APPLICATIONS OF CHUNKING

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Information Extraction
- Question Answering
- Extracting subcat frames
- providing additional features for ML methods



REPRESENTING CHUNKS: IOB TAGS



- Each token is tagged with one of three special chunk tags :
  `I` (inside), `O` (outside), `B` (begin)
- This format permits to represent more than one chunk type, so long as the chunks do not overlap.
- A token is tagged as `B` if it marks the beginning of a chunk.
- Subsequent tokens within the chunk are tagged `I`.
- All other tokens are tagged `O`.
- The `B` and `I` tags are suffixed with the chunk type, e.g. `B-NP`, `I-NP`.
- Is not necessary to specify a chunk type for tokens that appear outside a chunk, so these are just labeled `O`

CHUNKING WITH REGULAR EXPRESSIONS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Assume input is POS-tagged.

```
announce any new policy measures in his ...
VB       DT   JJ  NN     NNS      IN PRP$
```

- Identify chunks by sequences of tags

announce any new policy measures in his ...
VB **DT** **JJ** **NN** **NNS** IN PRP$

- Define rules in terms of tag patterns

  NP: {<DT><JJ><NN><NNS>}

  ....

RULE ORDERING

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- If a tag pattern matches at multiple overlapping locations, the first match takes precedence.
- For example, if we apply a rule that matches two consecutive nouns to a text containing three consecutive nouns, then the first two nouns will be chunked

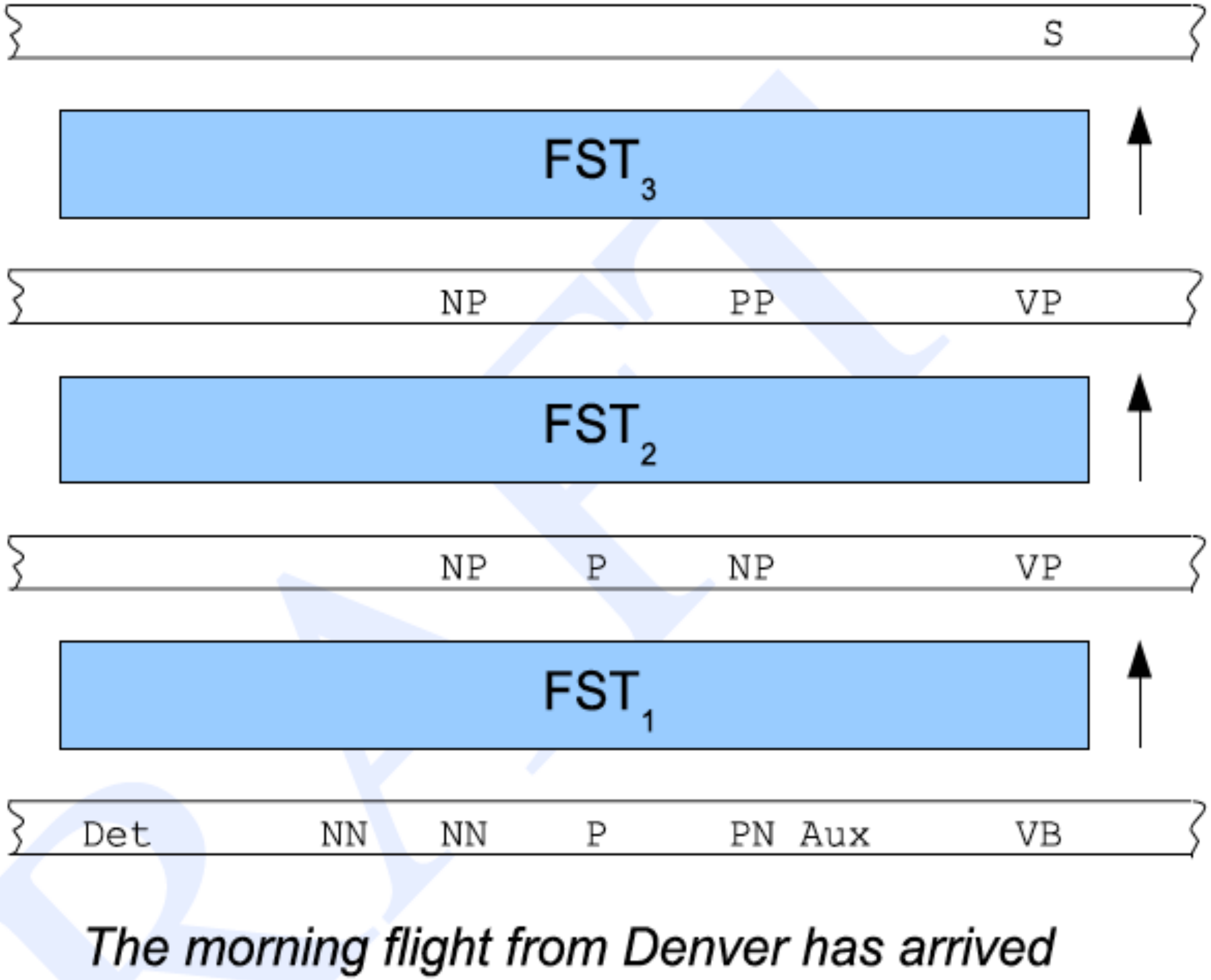
NP: {<NN><NN>}

```
money market fund        ➔        [money market] fund
NN    NN     NN
```

- Once we have created the chunk for money market, we have removed the context that would have permitted fund to be included in a chunk.

➔ order of rules is crucial. This is a general problem with rule-based systems

CASCADED TRANSDUCERS



Advantage:

- fast processing!

Disadvantage

- hard to define the FSTs in the right order
- maintenance and transferring of projects is time-consuming

DEVELOPMENT OF RULE-BASED SYSTEMS

- Development cycle: several iterations of creating/ changing rules and testing
- Tracing: infrastructure to mark, which rule was applied for particular chunks. This helps to identify problematic rules
- Hard to incorporate arbitrary features, need formalism to avoid inconsistencies
- The more rules already exist in the system, the harder it is to grasp their complex interactions

It is much more straightforward to use machine learning for sequence labeling. Of course, this requires a labeled training corpus.

SEQUENCE LABELING FOR CHUNKING

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- CoNLL-2000: Competition between systems to create the best chunker
- "Shared Task" setup:
  - training and validation data is public
  - test data is only known to the organizers
  - official evaluation, then test data is made public

Data Format:

```
Trust     NN    B-NP
in        IN    B-PP
the       DT    B-NP
pound     NN    I-NP
is        VBZ   B-VP
widely    RB    I-VP
expected  VBN   I-VP
to        TO    I-VP
take      VB    I-VP
another   DT    B-NP
sharp     JJ    I-NP
dive      NN    I-NP
```

count	%	type
55081	51%	NP (noun phrase)
21467	20%	VP (verb phrase)
21281	20%	PP (prepositional phrase)
4227	4%	ADVP (adverb phrase)
2207	2%	SBAR (subordinated clause)
2060	2%	ADJP (adjective phrase)
556	1%	PRT (particles)
56	0%	CONJP (conjunction phrase)
31	0%	INTJ (interjection)
10	0%	LST (list marker)
2	0%	UCP (unlike coordinated phrase)

EVALUATION OF CHUNKING

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- With IOB-representation, we can look at
  - single label accuracy: Per I/O/B label
  - chunk precision: is an identified chunk correct?
  - chunk recall: how many of all chunks did the system find correctly?
- IR-inspired measures:

$$\text{Precision } P = \frac{\text{number of correctly identified chunks}}{\text{total number of chunks returned}} = \frac{tp}{tp+fp}$$

$$\text{Recall } R = \frac{\text{number of correctly identified chunks}}{\text{total number of chunks in test set}} = \frac{tp}{tp+fn}$$

$$F1 = \frac{2 \cdot P \cdot R}{P+R} \text{ is harmonic mean of } P \text{ and } R$$

		Test Data	
		chunk	not chunk
System response	chunk	**tp** (true positive)	**fp** (false positive)
	not chunk	**fn** (false negative)	**tn** (true negative)

RESULTS OF CONLL-2000 CHUNKING EVALUATION

test data	precision	recall	$F_{\beta=1}$
Kudoh and Matsumoto	93.45%	93.51%	93.48
Van Halteren	93.13%	93.51%	93.32
Tjong Kim Sang	94.04%	91.00%	92.50
Zhou, Tey and Su	91.99%	92.25%	92.12
Déjean	91.87%	92.31%	92.09
Koeling	92.08%	91.86%	91.97
Osborne	91.65%	92.23%	91.94
Veenstra and Van den Bosch	91.05%	92.03%	91.54
Pla, Molina and Prieto	90.63%	89.65%	90.14
Johansson	86.24%	88.25%	87.23
Vilain and Day	88.82%	82.91%	85.76
baseline	72.58%	82.14%	77.07

- Baseline: Most frequent label per POS tag
- Best systems now use Bi-LSTM or Bi-GRUs

SYNTACTIC PARSING WITH CFGS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Recap: A grammar $G = (\Phi,\Sigma,R,S)$ is **context-free**, iff all production rules in *R* obey the form $A \rightarrow \alpha$ with $A \in \Phi$, $\alpha \in (\Phi \cup \Sigma)^*$.

Grammar Rules

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

Lexicon

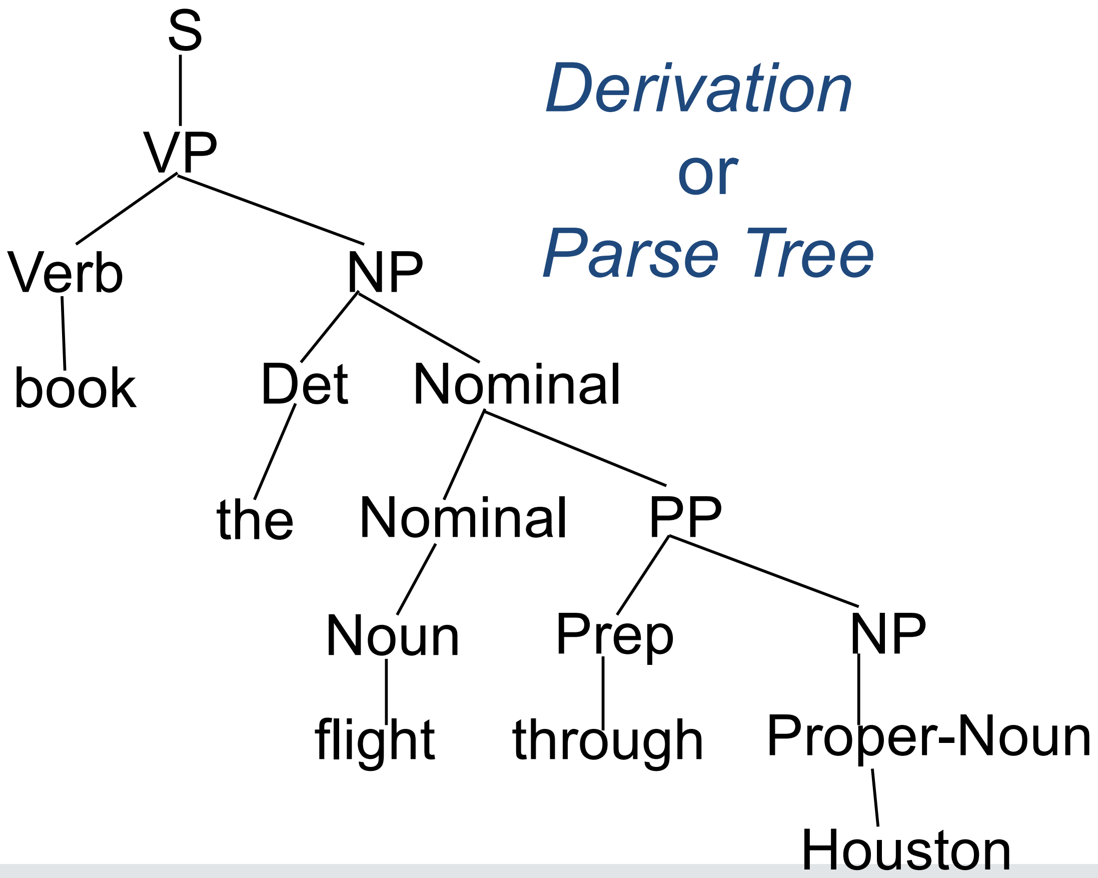
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

SENTENCE GENERATION

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminal symbols remain



Derivation
or
Parse Tree

WHY CAN'T WE USE FSAS?

- Language is left/right recursive:
  - {tall, green, slimy, calm, …}* frog
  - the house {with a roof, with a door, with a window, with a garden, …}*
- Can process these recursions with FSAs: ADJ* NN , NN (P DET NN)*

- But language has also center-embedded recursions:
  - the door opens
  - the door that uncle Henry repaired opens
  - the door that uncle Henry who the dog bit repaired opens
  - the door that uncle Henry who the dog that John owns bit repaired opens
  - …
  - (this is even more fun in German)
- Center-embedding recursion is not regular, need tree structure!

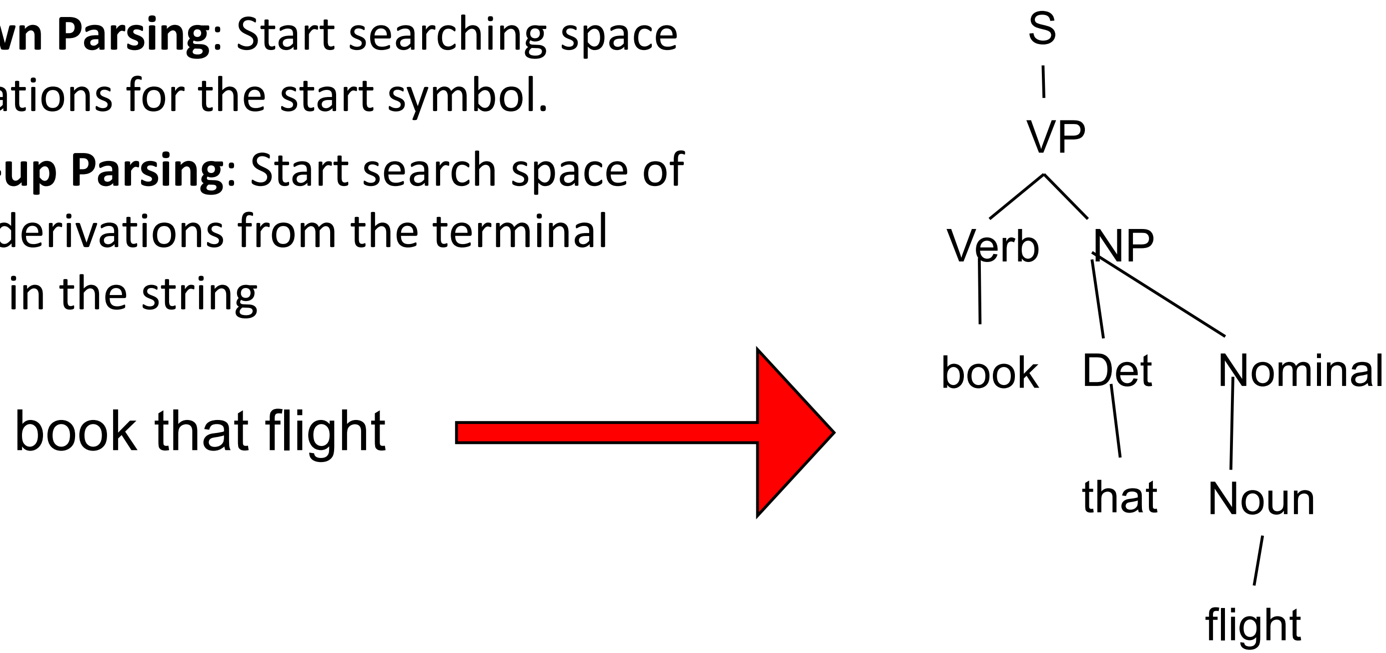
Karlsson, Fred. (2007). Constraints on multiple center-embedding of clauses. Journal of Linguistics 43 (2): 365-392.

PARSING:
BOTTOM UP VS. TOP-DOWN

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux→ does

- Given a string of terminals and a CFG, determine if the string can be generated by the CFG.
  - Also return a parse tree for the string
  - Also return all possible parse trees for the string
- Must search space of derivations for one that derives the given string.
  - **Top-Down Parsing**: Start searching space of derivations for the start symbol.
  - **Bottom-up Parsing**: Start search space of reverse derivations from the terminal symbols in the string

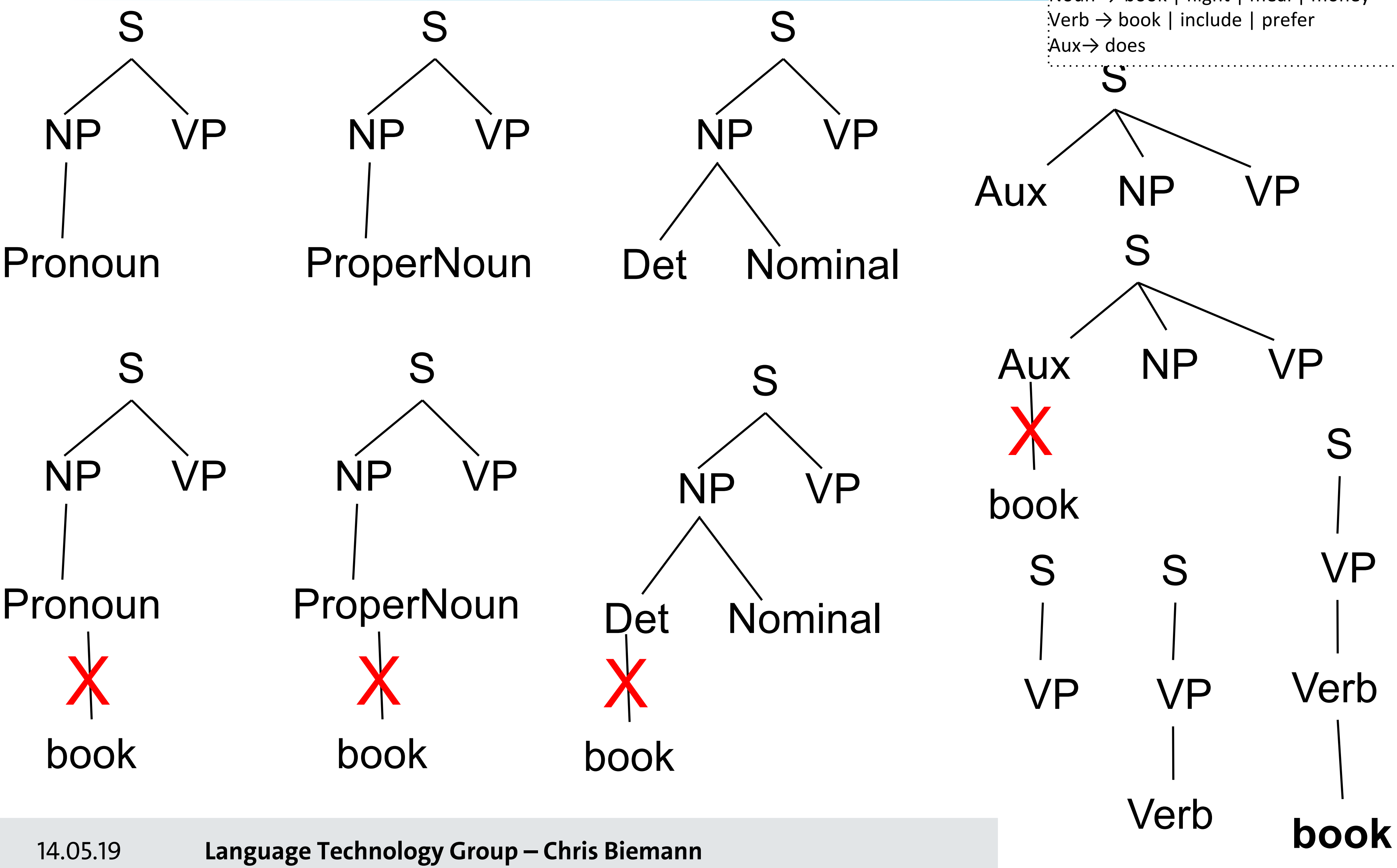


Example by Ray Mooney, UT at Austin

TOP DOWN PARSING: BOOK THAT FLIGHT

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

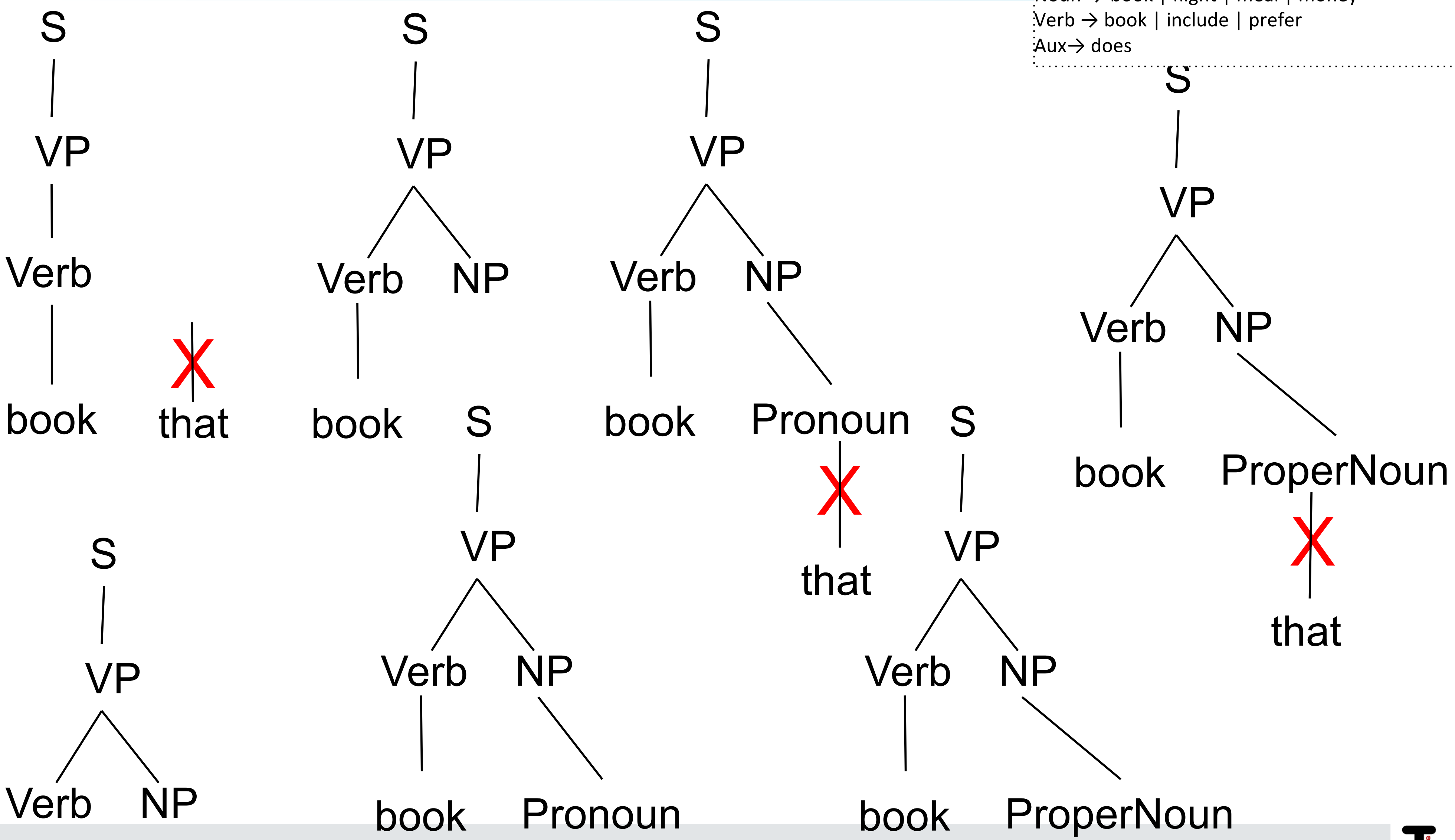
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux→ does



TOP DOWN PARSING:
BOOK THAT FLIGHT

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

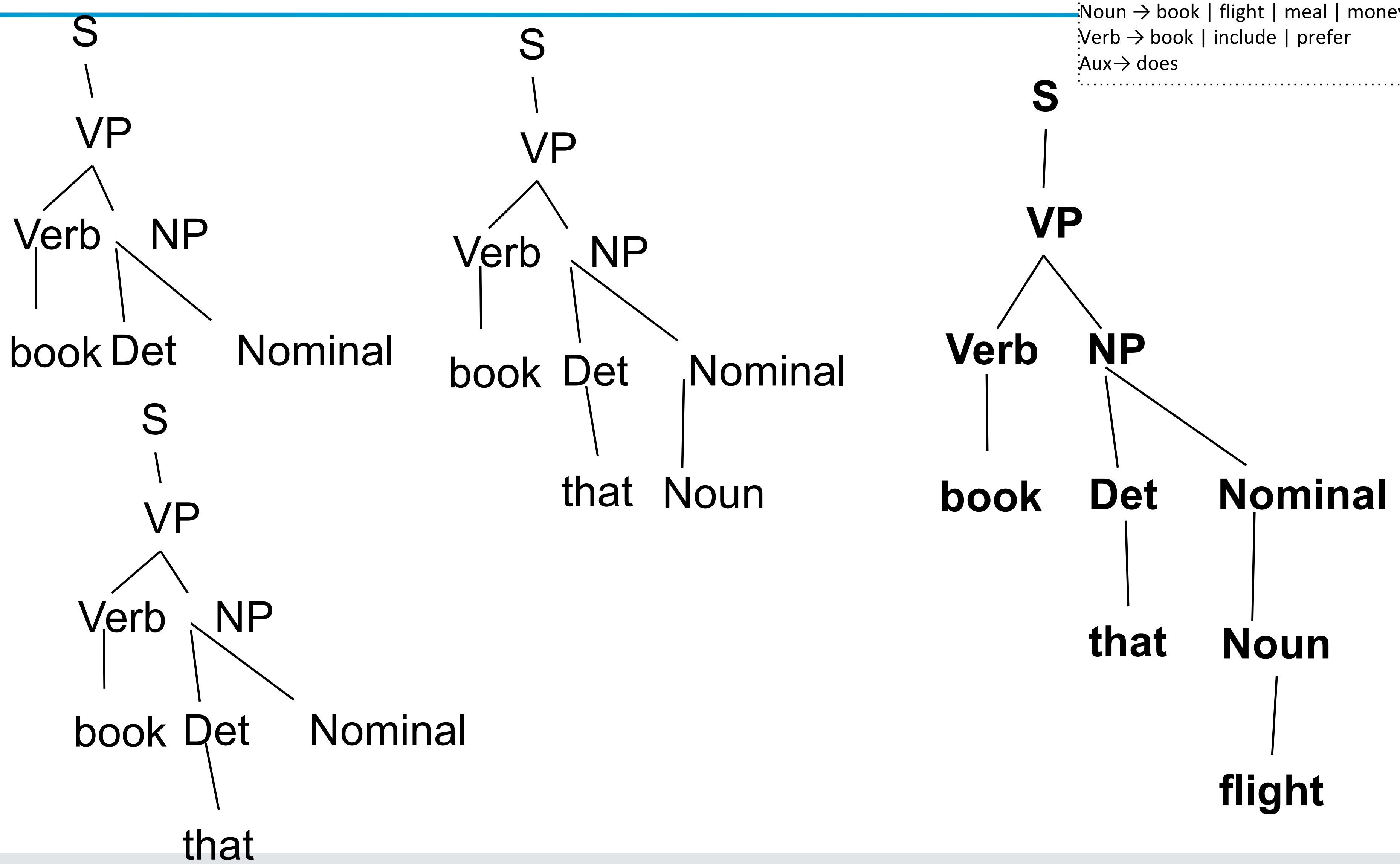
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux→ does



TOP DOWN PARSING: BOOK THAT FLIGHT

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

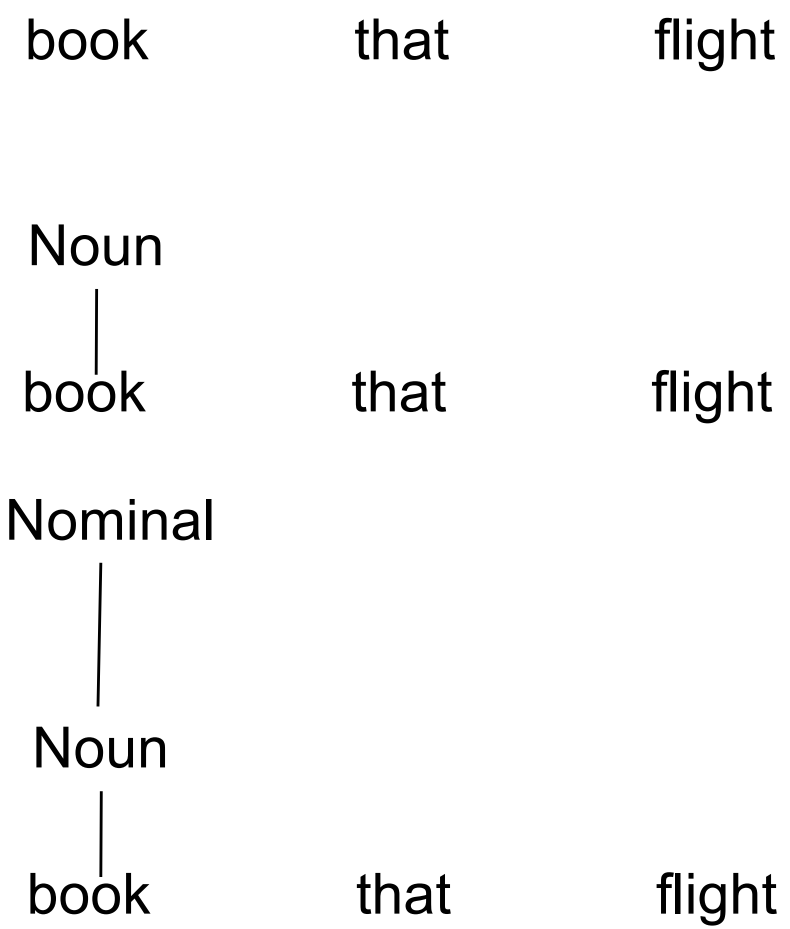
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux→ does

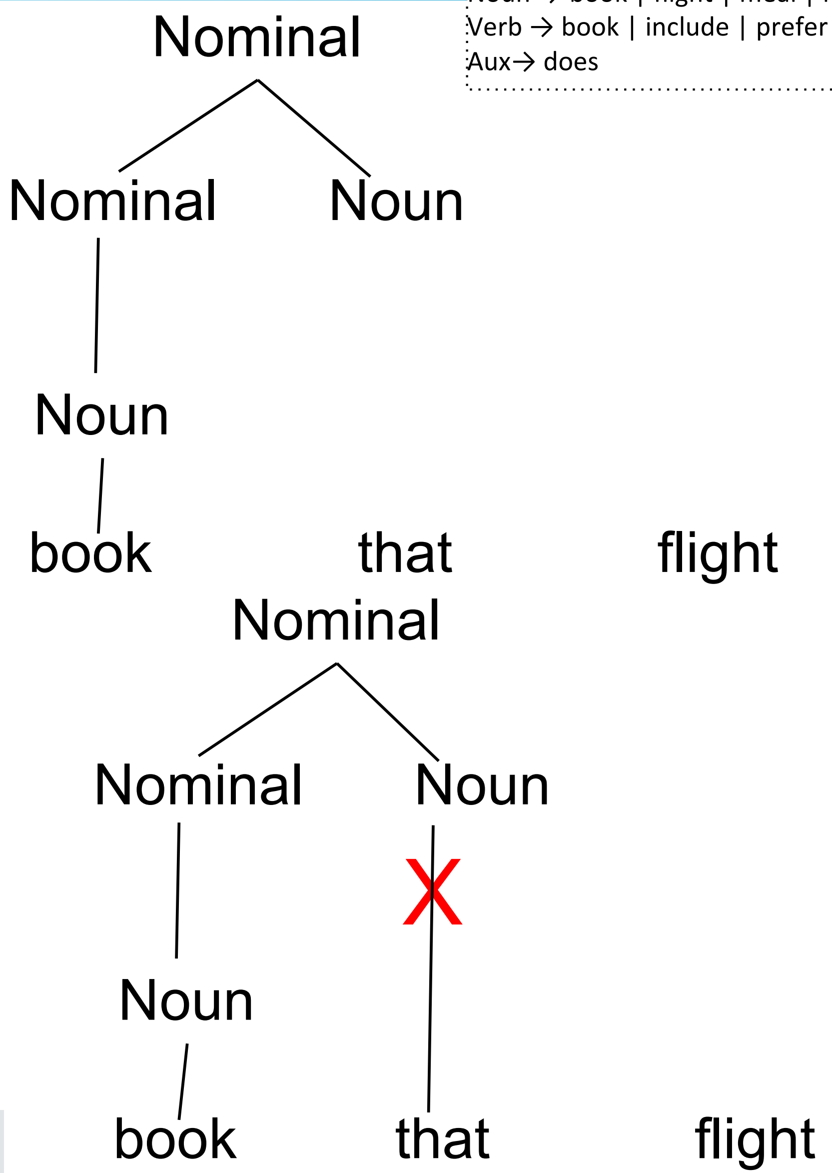


BOTTOM UP PARSING: BOOK THAT FLIGHT

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux → does

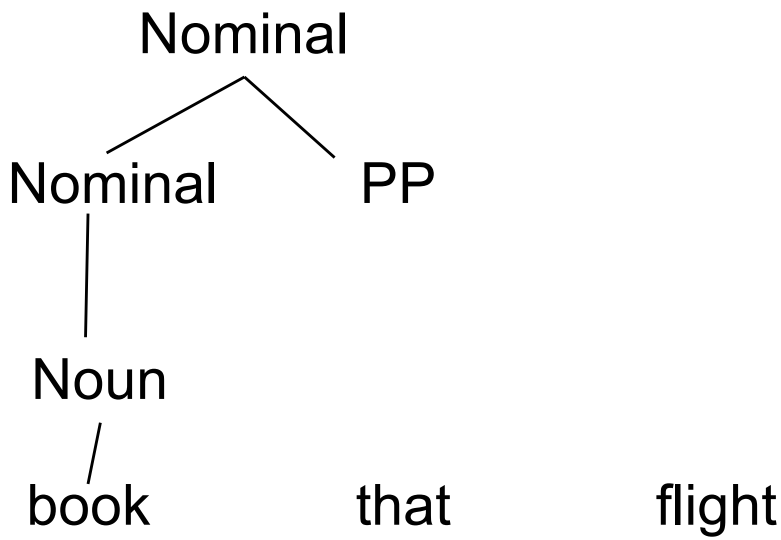


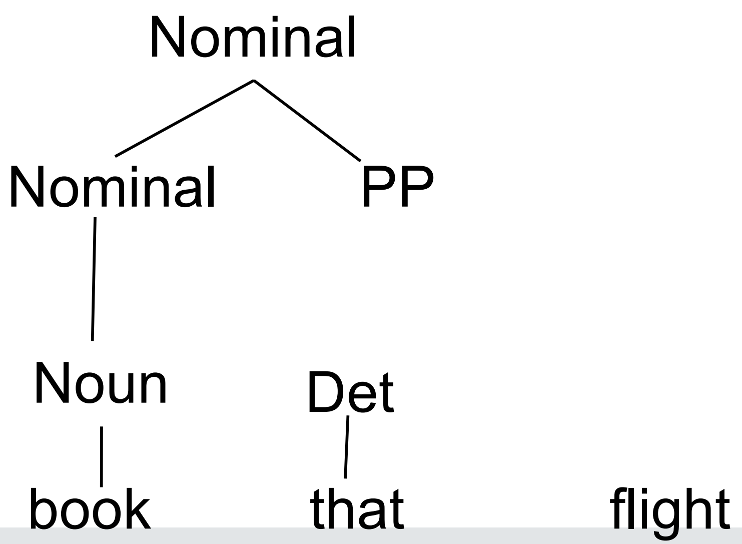


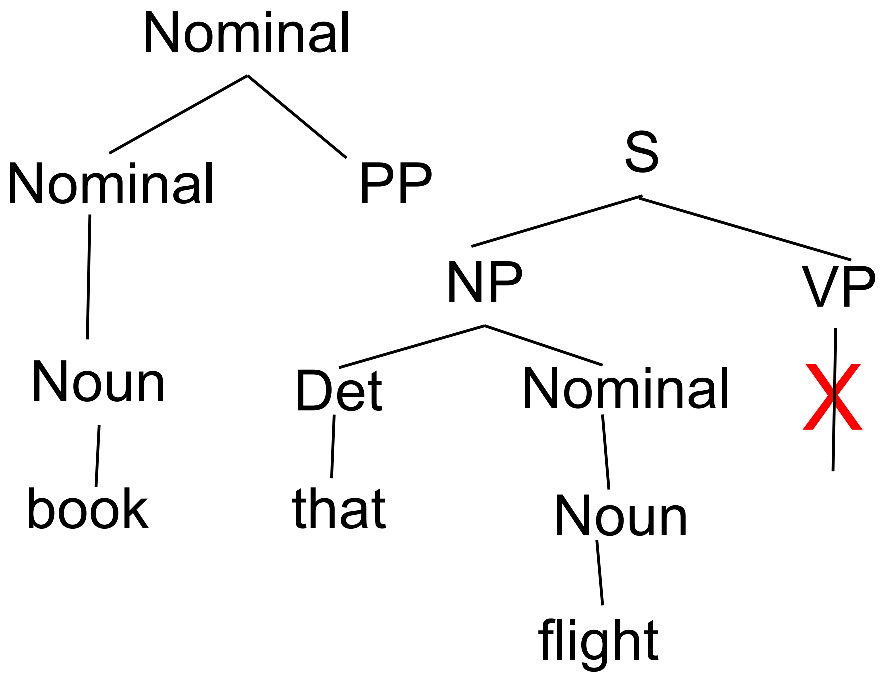
BOTTOM UP PARSING: BOOK THAT FLIGHT

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux→ does



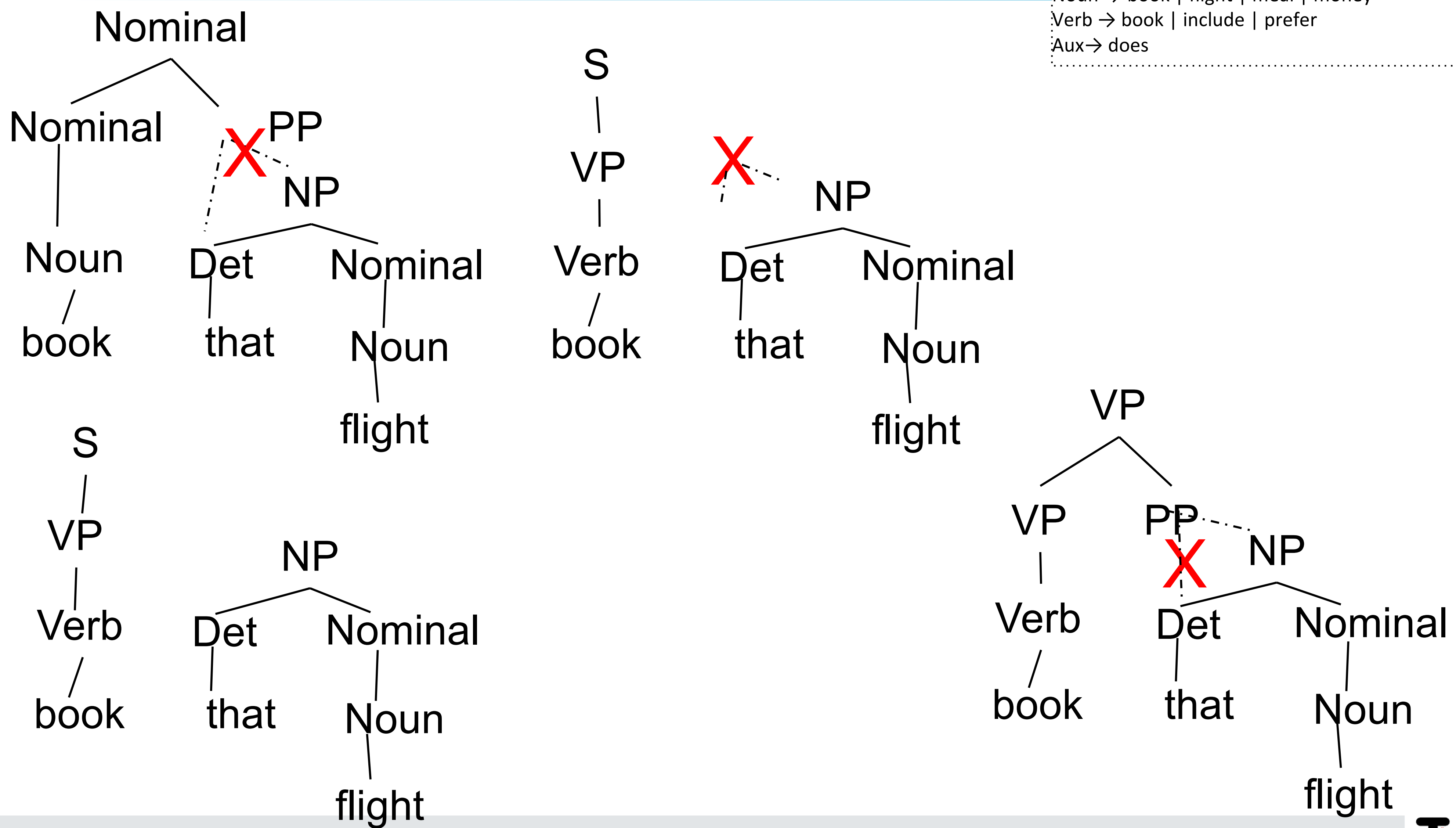




BOTTOM UP PARSING: BOOK THAT FLIGHT

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

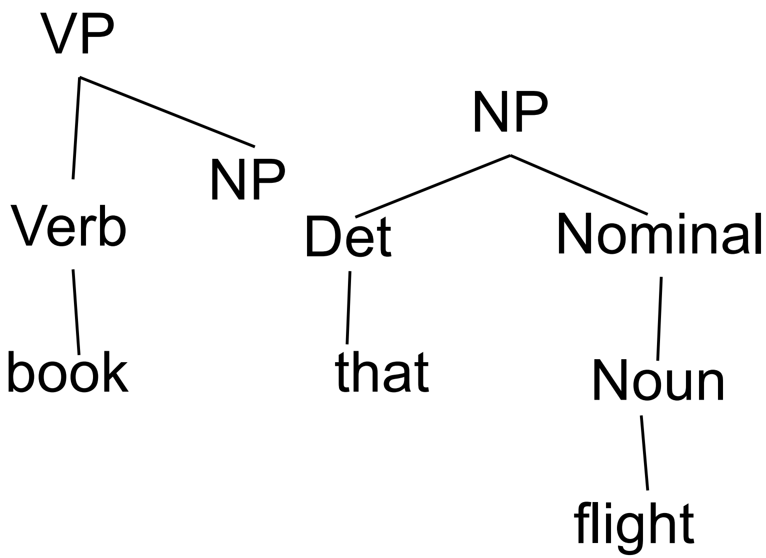
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux→ does

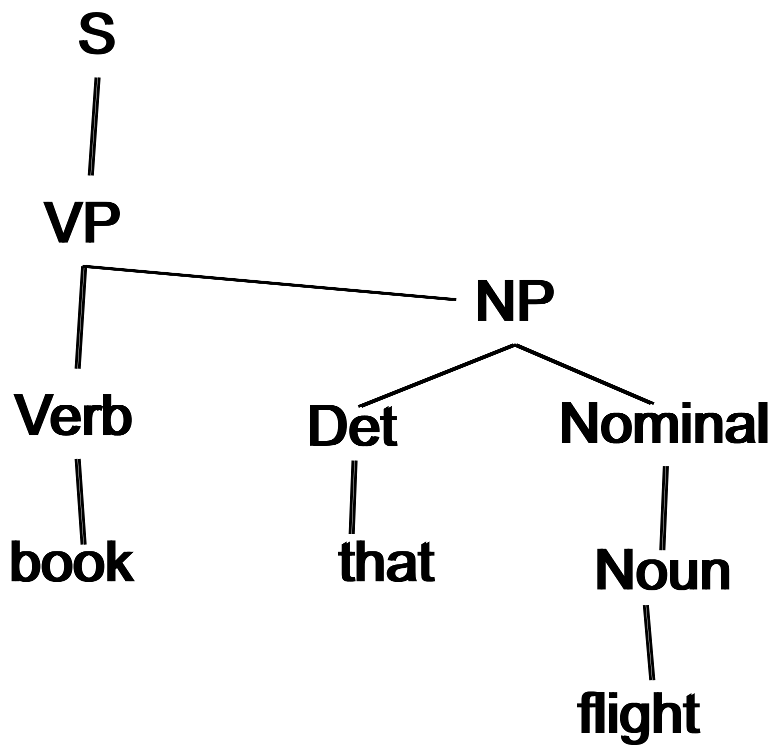


BOTTOM UP PARSING: BOOK THAT FLIGHT

S → NP VP | Aux NP VP | VP
NP → Pronoun | Proper-Noun | Det Nominal
Nominal → Noun | Nominal Noun | Nominal PP
VP → Verb | Verb NP | VP PP
PP → Prep NP

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Aux→ does





TOP DOWN VS. BOTTOM UP

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.
- Complexity of naïve implementation: Exponential due to branching

DYNAMIC PROGRAMMING PARSING

- To avoid extensive repeated work, must cache intermediate results, i.e. completed sub-phrases.
- Caching (memorizing) critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.
- Dynamic programming algorithms based on both top-down and bottom-up search can achieve $O(n^3)$ recognition time where n is the length of the input string.
- NB: Parsing methods for CFGs are similar for programming languages and natural language

DYNAMIC PROGRAMMING PARSING METHODS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- **CYK** (Cocke-Younger-Kasami, 1967) algorithm based on bottom-up parsing and requires first normalizing the grammar.
- **Earley** (1970) parser is based on top-down parsing, does not require normalizing the grammar, but is more complex.
- More generally, **chart parsers** retain completed phrases in a chart and can combine top-down and bottom-up search.
- Complexity of $O(n^3)$ can further be improved for certain grammars, i.e. unambiguous grammars – however, not for interesting grammars of natural language

EARLEY PARSING

- Dynamic Programming solution for **top-down** parsing
- Allows arbitrary CFGs
- Fills a table in a single sweep over the input words
- Table is length N+1; N is number of words
- **States** (table entries) represent:
  - **Completed** constituents and their locations
  - **In-progress** constituents
  - **Predicted** constituents

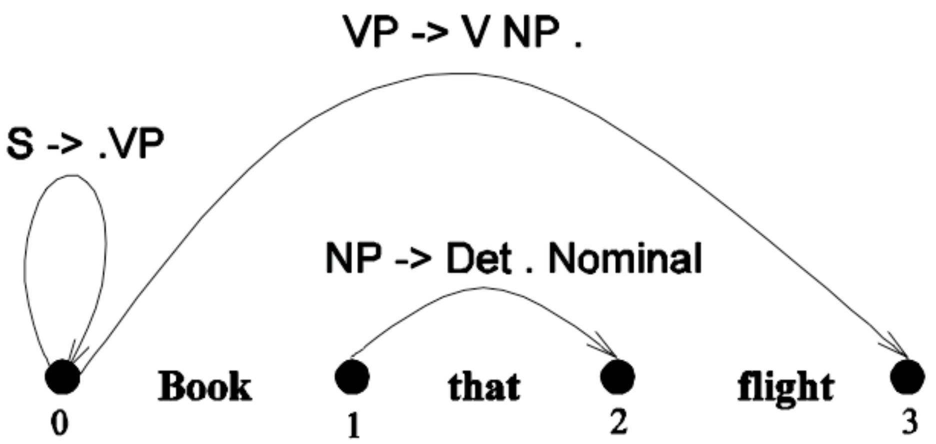
STATES AND LOCATIONS

The table entries are called states and are represented with dotted rules:

S -> • VP	A VP is predicted
NP -> Det • Nominal	An NP is in progress
VP -> V NP •	A VP has been found

With offsets:

S -> • VP [0,0]	A VP is predicted at the start of the sentence
NP -> Det • Nominal [1,2]	An NP is in progress; the Det goes from 1 to 2
VP -> V NP • [0,3]	A VP has been found starting at 0 and ending at 3



EARLEY PARSING

- Successful parse: S state in the final column that spans from 0 to n and is complete: S –> α • [0,n]
- Sweep through the table from 0 to n:
  – New predicted states are created by starting top-down from S
  – New incomplete states are created by advancing existing states as new constituents are discovered
  – New complete states are created in the same way.
- More specifically:

1. Predict all the states you can upfront
2. Read a word
   1. Extend states based on matches
   2. Generate new predictions
   3. repeat until the end of the sentence
3. Look at n to see if you have a winner

EARLEY ALGORITHM

```
function EARLEY-PARSE(words, grammar) returns chart

  ADDTOCHART((γ → • S, [0,0]), chart[0])
  for i ← from 0 to LENGTH(words) do
   for each state in chart[i] do
     if INCOMPLETE?(state) and
              NEXT-CAT(state) is not a part of speech then
        PREDICTOR(state)
     elseif INCOMPLETE?(state) and
              NEXT-CAT(state) is a part of speech then
        SCANNER(state)
     else
        COMPLETER(state)
   end
  end
  return(chart)
```

```
procedure PREDICTOR((A → α • B β, [i, j]))
   for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
      ADDTOCHART((B → • γ, [j, j]), chart[j])
   end

procedure SCANNER((A → α • B β, [i, j]))
   if B ∈ PARTS-OF-SPEECH(word[j]) then
      ADDTOCHART((B → word[j] •, [j, j+1]), chart[j+1])

procedure COMPLETER((B → γ •, [j,k]))
   for each (A → α • B β, [i, j]) in chart[j] do
      ADDTOCHART((A → α B • β, [i,k]), chart[k])
   end
```

EXAMPLE: BOOK THAT FLIGHT

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Chart[0]	S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor

EXAMPLE: BOOK THAT FLIGHT

Chart[1]	S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner
	S13	$VP \rightarrow Verb \bullet$	[0,1]	Completer
	S14	$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
	S15	$VP \rightarrow Verb \bullet NP\ PP$	[0,1]	Completer
	S16	$VP \rightarrow Verb \bullet PP$	[0,1]	Completer
	S17	$S \rightarrow VP \bullet$	[0,1]	Completer
	S18	$VP \rightarrow VP \bullet PP$	[0,1]	Completer
	S19	$NP \rightarrow \bullet Pronoun$	[1,1]	Predictor
	S20	$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor
	S21	$NP \rightarrow \bullet Det\ Nominal$	[1,1]	Predictor
	S22	$PP \rightarrow \bullet Prep\ NP$	[1,1]	Predictor

EXAMPLE: BOOK THAT FLIGHT

Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
	S24	*NP* → *Det* • *Nominal*	[1,2]	Completer
	S25	*Nominal* → • *Noun*	[2,2]	Predictor
	S26	*Nominal* → • *Nominal Noun*	[2,2]	Predictor
	S27	*Nominal* → • *Nominal PP*	[2,2]	Predictor
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	Completer
	S30	*NP* → *Det Nominal* •	[1,3]	Completer
	S31	*Nominal* → *Nominal* • *Noun*	[2,3]	Completer
	S32	*Nominal* → *Nominal* • *PP*	[2,3]	Completer
	S33	*VP* → *Verb NP* •	[0,3]	Completer
	S34	*VP* → *Verb NP* • *PP*	[0,3]	Completer
	S35	*PP* → • *Prep NP*	[3,3]	Predictor
	S36	*S* → *VP* •	[0,3]	Completer
	S37	*VP* → *VP* • *PP*	[0,3]	Completer

NOTES ON EARLEY

Participating states of the final parse:

Chart[1]	S12	*Verb* → *book* •	[0,1]	Scanner
Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	Completer
	S30	*NP* → *Det Nominal* •	[1,3]	Completer
	S33	*VP* → *Verb NP* •	[0,3]	Completer
	S36	*S* → *VP* •	[0,3]	Completer

- For such a simple example, there is a lot of 'useless' steps
- Earley predicts next constituents that are not consistent with the input
- Possible to improve the algorithm by look-ahead strategies

CYK PARSING ALGORITHM

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- First, grammar must be converted to Chomsky normal form (CNF) in which productions must have either exactly 2 non-terminal symbols on the RHS or 1 terminal symbol (lexicon rules).
- Parse bottom-up, storing phrases formed from all substrings in a triangular table (chart).
- Parse trees are for CNF grammar, not the original grammar.
- A post-process can repair the parse tree to return a parse tree for the original grammar.

CONVERSION TO CHOMSKY NORMAL FORM

1. Introduce a new start symbol S_0, add rule $S_0 \rightarrow S$ (S=old start symbol)
2. Eliminate all ε rules of the form $A \rightarrow \varepsilon$ ($A \neq S_0$): remove rule and split rules containing A on the RHS in all versions, with and without A's. For rules $B \rightarrow A$, replace A with ε if B has not been through this step yet, otherwise eliminate $B \rightarrow A$.
3. Eliminate all unit rules $A \rightarrow B$, by adding all $B \rightarrow R_i$ to $A \rightarrow R_i$ where R_i is not a unit rule. If R_i is a unit rule add all $R_i \rightarrow K_i$ to A ($A \rightarrow K_i$) where K_i is not a unit rule. Continue this process for all following unit-rules, until we observe a unit rule we have seen in the cleaning step. Then eliminate $A \rightarrow B$.
4. Clean up remaining rules: For $A \rightarrow R_1, R_2, .. R_n$ ($n>2$, R_i terminals or non-terminals), create a chain $\{A \rightarrow R_1\ A_1, A_1 \rightarrow R_2\ A_2 \ldots A_{n-2} \rightarrow R_{n-1}\ R_n\}$. For all R_i that are terminals, create a lexicon rule and replace R_i with its LHS.
5. If $S_0 \rightarrow C$ remains, set C as start symbol.

EXAMPLE CONVERSION TO CNF

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Original grammar

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

Grammar in CNF

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

CYK PARSER

	Book	the	flight	through	Houston
	j= 1	2	3	4	5
i= 0					
1					
2					
3					
4					

Cell[*i*,*j*] contains all constituents (non-terminals) covering words *i* +1 through *j*

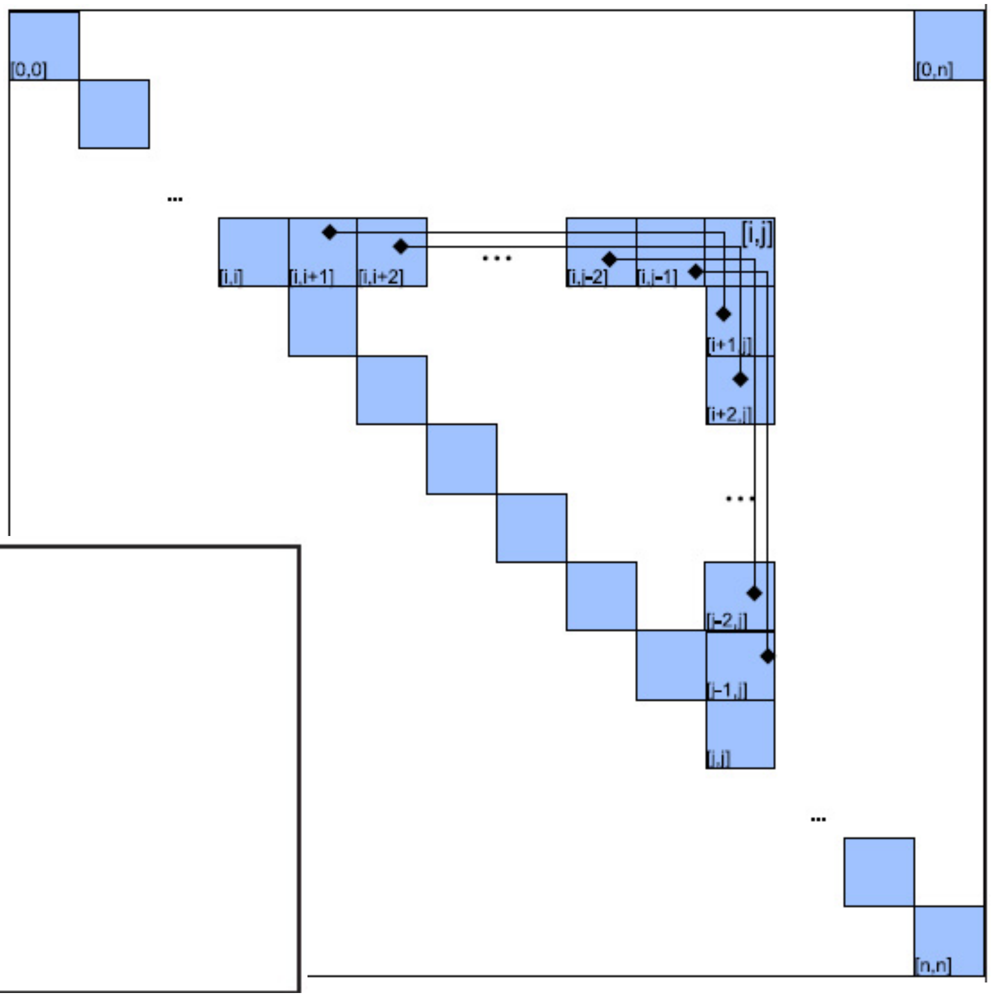
CYK ALGORITHM

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid A \rightarrow BC \in grammar,$
          $B \in table[i,k],$
          $C \in table[k,j]\}$

CYK SEARCH SPACE

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid A \rightarrow BC \in grammar,$
          $B \in table[i,k],$
          $C \in table[k,j]\}$

CHART PARSING (KAY, 1982)

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- combines (some) advantages of bottom-up and top-down
- CYK and Earley are special cases of the more general chart parsing scheme
- Both top-down and bottom-up steps are carried out in the order of an **agenda**, which is updated dynamically.
- **fundamental rule of chart parsing**: when the chart contains two contiguous edges (states) where one provides the constituent for the other, then they should be combined in the next step

Chart parsing does not lower the worst-case complexity but requires lower constants on average by making smart choices about the next step.

LIMITS OF CFGS FOR NATURAL LANGUAGE PARSING

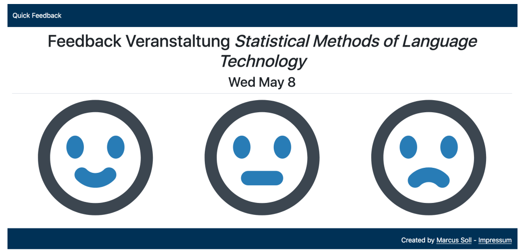
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Ambiguity resolution is not handled: just produces all possible parse trees
- Addressing some grammatical constraints requires complex CFGs that do no compactly encode the given regularities.
- Some aspects of natural language syntax may not be captured at all by CFGs and require context-sensitivity (productions with more than one symbol on the LHS)
- Agreement handling is painful:
  - Subjects must agree with their verbs on person and number
  - gender agreement
  - case agreement
  - ➔ need to split production rules as to account for these effects
- Subcategorization: Verbs take only some types of arguments, but not others
  - E.g. wrong subcategorization: John found. John disappeared the ring.

CONCLUSIONS ON PARSING WITH CFGS

- Syntax parse trees specify the syntactic structure of a sentence that helps determine its meaning.
  - John ate the spaghetti with meatballs with chopsticks.
  - How did John eat the spaghetti? What did John eat?
- CFGs can be used to define the grammar of a natural language.
- Dynamic programming algorithms allow computing a single parse tree in cubic time or all parse trees in exponential time.

IMMEDIATE FEEDBACK



- Jurafsky, D. and Martin, J. H. (2009): Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Second Edition. Pearson: New Jersey: Chapter 14
- Manning, C. D. and Schütze, H. (1999): Foundation of Statistic atural Language Processing. MIT Press: Cambridge, M husett ter
- with further examples by Ray Mooney, U

coming up next

PCFGs, probabilistic CYK, dependen pars

STATISTICAL PARSING