Lecture 8: Constituency and Dependency Parsing

Suchir Salhan

Faculty of Modern and Medieval Languages and Linguistics

Li18 Computational Linguistics

Theories of syntax

We're looking at a theory of syntax (**Context Free Grammars**) based on the ideas of constituency and phrase structures.

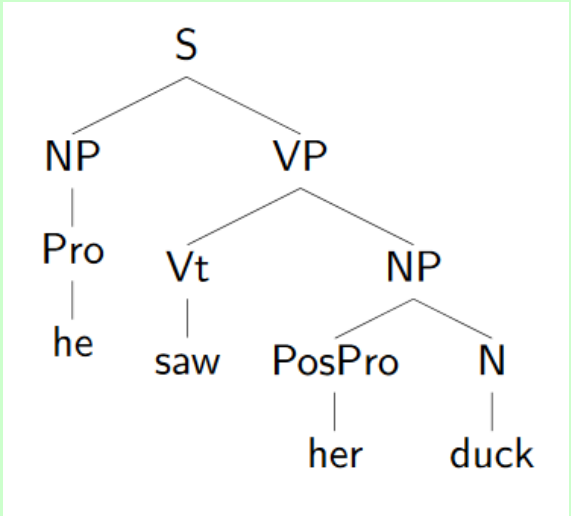
Be aware that there is an alternative view of linguistic structure (**Dependency grammars**) that is covered briefly in J&M-2 (Section 12.7) and as a full chapter in J&M-3.

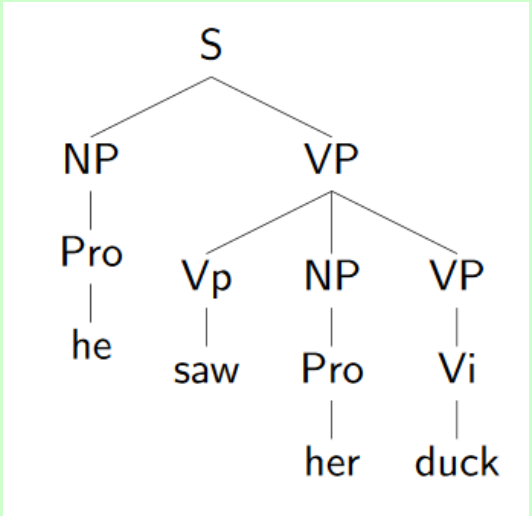
For CFGs, as with other models, we're considering:

- What CFGs can capture, and what they cannot.
- Algorithms that provide syntactic analysis for sentences.

A quick reminder: structural ambiguity

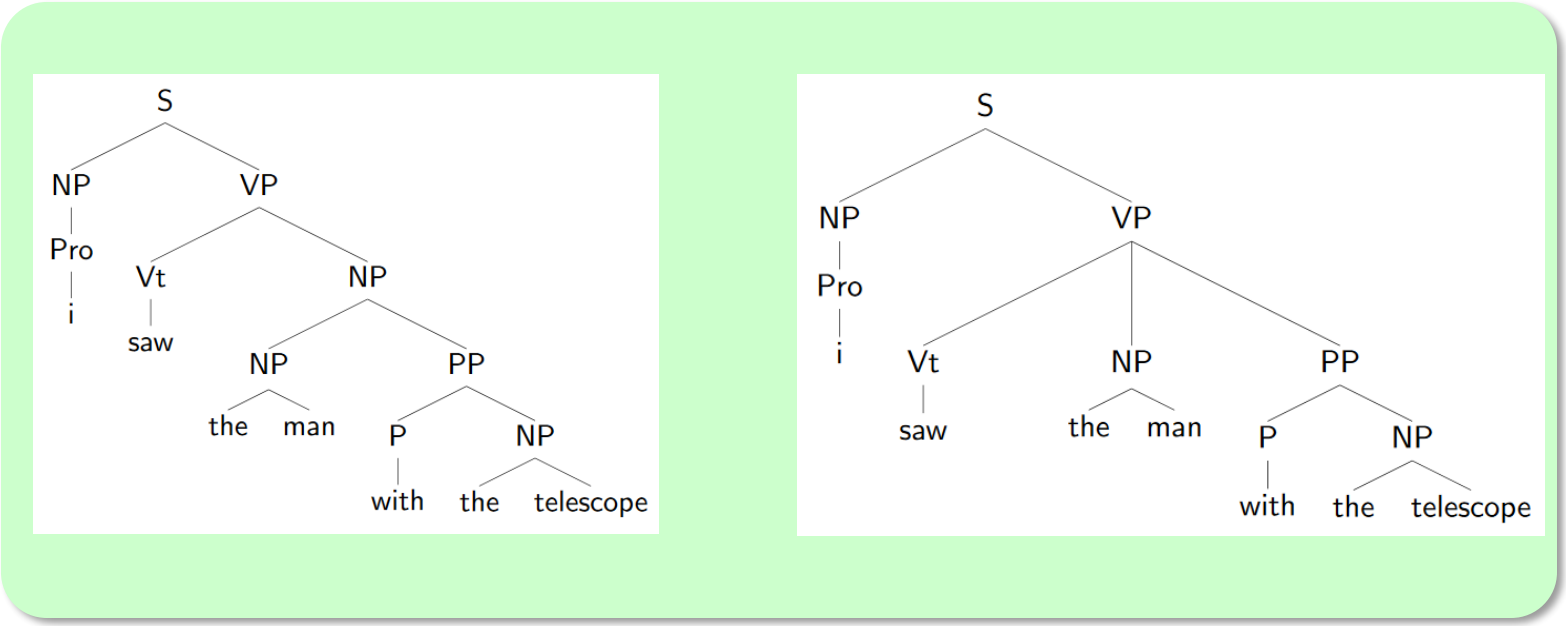
In the homework for your 3rd supervision you discussed the issue that some sentences have more than one parse: structural ambiguity.





Example of structural ambiguity caused by POS ambiguity

A quick reminder: structural ambiguity



- Example of structural ambiguity caused by attachment ambiguity

A quick reminder: structural ambiguity

the old men and women

[the [old men] and [women]]

[the [old [men and women]]]

- Example of structural ambiguity caused by coordination ambiguity

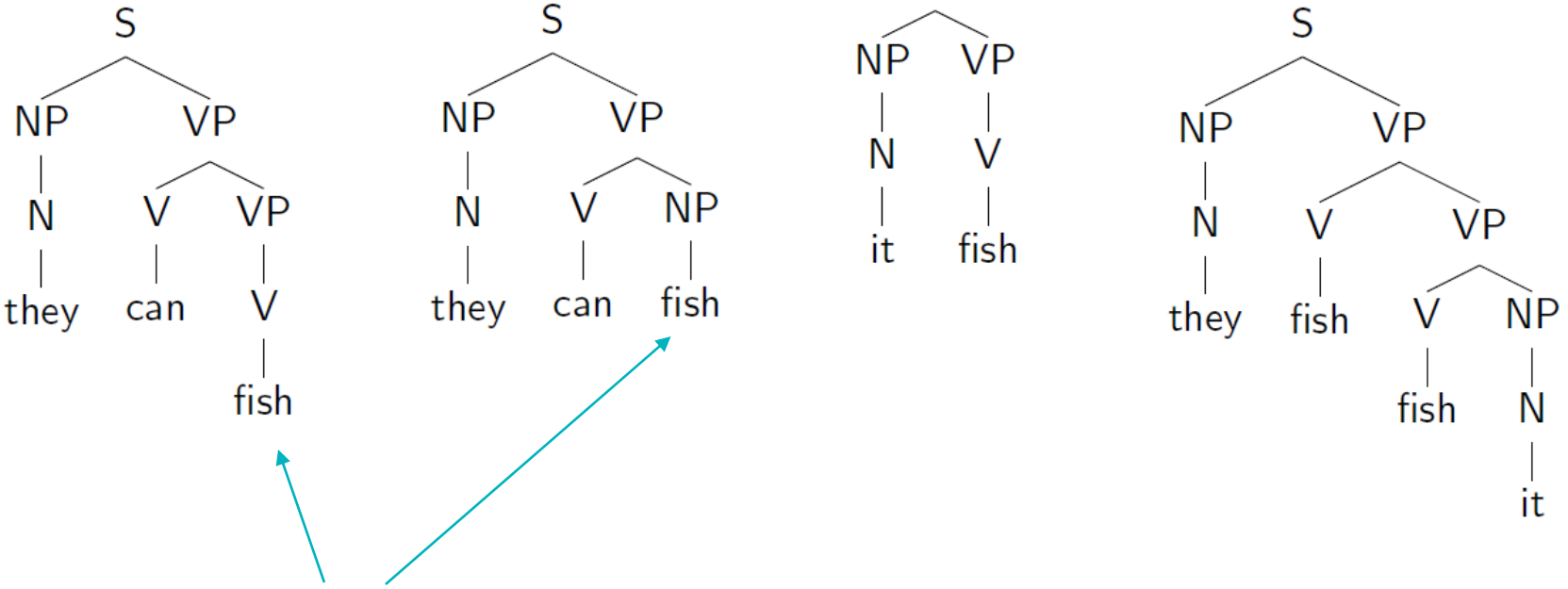
Summary

1. **A toy CFG**
2. Charts
3. Earley parsing (Top-Down)
4. Chart parsing (Bottom-Up, CKY Algorithm)
5. Parsing and Language Processing (Incremental Parsing)
6. Resolving Multiple Parses
7. Dependency Parsing

CFG example

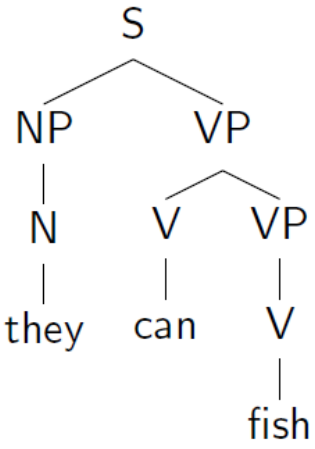
Start Symbol	$\mathcal{S}$		
Terminal symbols	Σ	=	{can, fish, rivers, pools, December, Scotland, they, in}
Non-terminal symbols	$\mathcal{N}$	=	{NP, VP, PP, N, V, P}
Rewrite rules	$\mathcal{R}$	=	S $\rightarrow$ NP VP
			NP $\rightarrow$ N PP
			NP $\rightarrow$ N
			PP $\rightarrow$ P NP
			VP $\rightarrow$ VP PP
			VP $\rightarrow$ V VP
			VP $\rightarrow$ V NP
			VP $\rightarrow$ V
			N $\rightarrow$ {it,fish,rivers,pools,December,Scotland,they}
			P $\rightarrow$ {in}
			V $\rightarrow$ {can,fish}

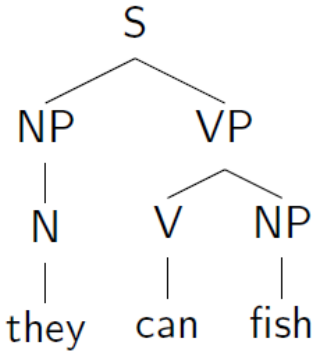
Some issued raised by our CFG

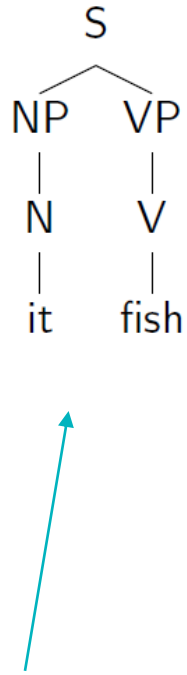


Structural ambiguity caused by POS ambiguity

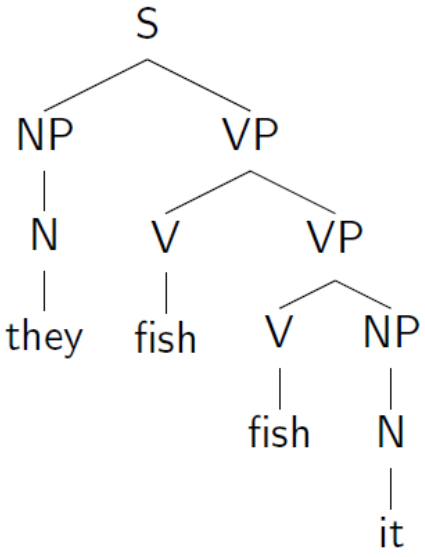
Some issued raised by our CFG



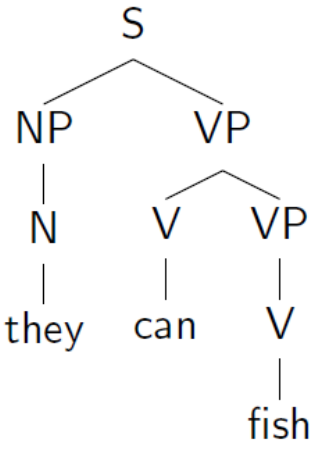


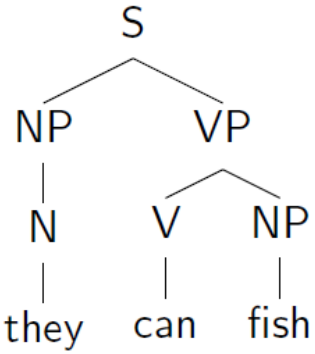


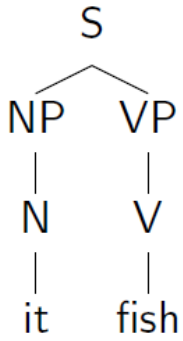
Subject-verb agreement violation

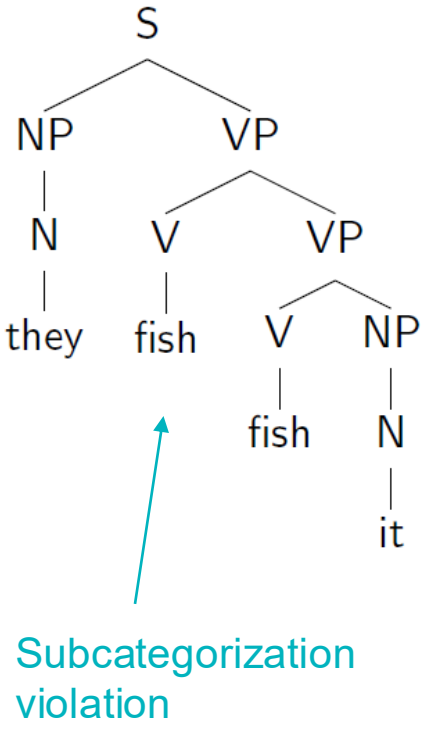


Some issued raised by our CFG









Constituency Parsing (Parsing CFGs): Where we are going today?

How can we deterministically recognise natural languages in a single derivation without backtracking?

All Context-Free Languages (including those exhibiting ambiguity) can be recognised in polynomial time using **dynamic programming algorithms**.

We will be exploring classical parsing algorithms for **constituency parsing.**

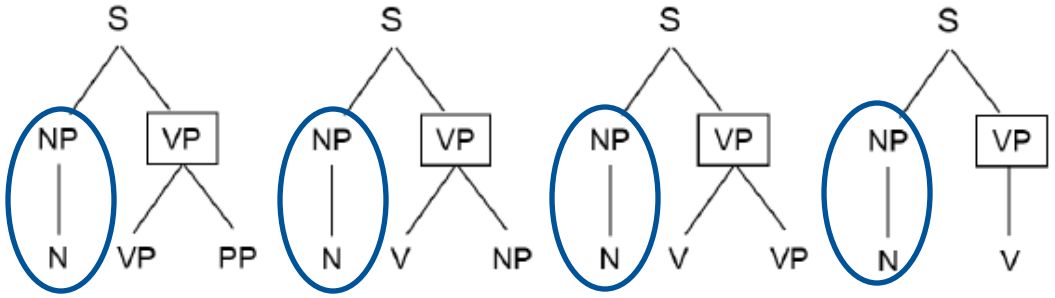
- **Chart Parsing:** A **chart parser** is any parser that:
  - Uses a **chart** (a data structure storing partial parses), and
  - uses **dynamic programming** to avoid recomputing substructures.
  - CKY/CYK, a **bottom-up chart parsing algorithm**, and the Earley Parser

Summary

1. A toy CFG
2. **Charts**
3. Earley parsing (Top-Down)
4. Chart parsing (Bottom-Up, CKY Algorithm)
5. Parsing and Language Processing (Incremental Parsing)
6. Resolving Multiple Parses
7. Dependency Parsing

Parser efficiency: avoiding repeated effort

When we parse there is often a lot of overlap in the trees:



We should avoid re-analysing any substring because its analysis is independent from the rest of the parse → Re-using tree parts wherever possible will save time.

We can use a data structure called a Chart to store all the things we have seen.

Charts

Chart parsing algorithms exploit the re-use of substrings in order to save time.

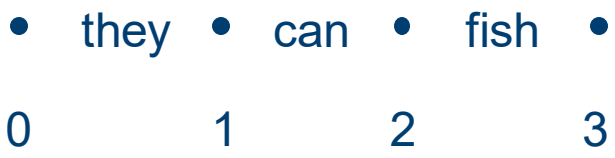
- Use dynamic programming to store and re-use sub-parses, then compose them into a full solution.
- Multiple potential parses can be explored at once.

Charts

A chart contains many Edges which record information regarding bits of the tree (or potential bits of tree).

An edge has a Span, indicating which words in the sentence this bit of tree relates to.

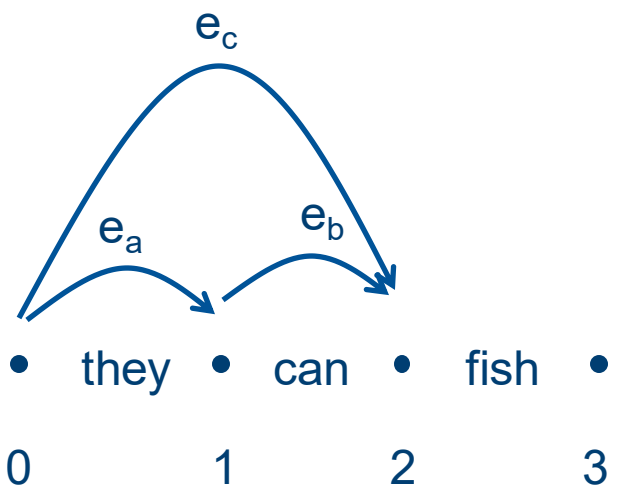
$edge_n$	Span



Charts

A chart contains many Edges which record information regarding bits of the tree (or potential bits of tree).

An edge has a Span, indicating which words in the sentence this bit of tree relates to.

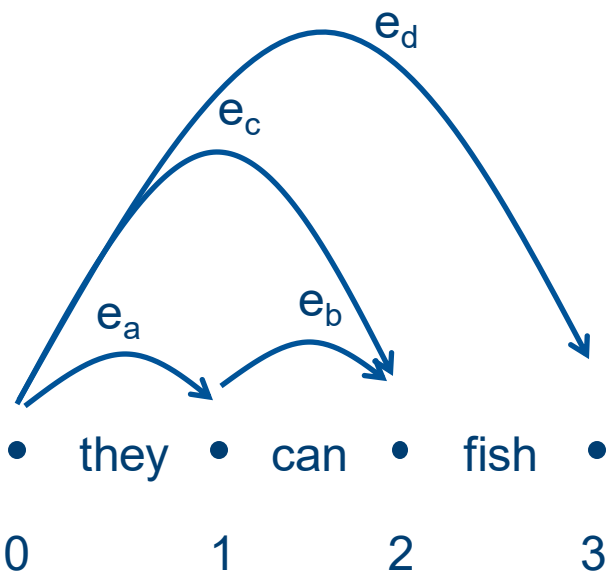


$edge_n$	Span
e_a	[0,1]
e_b	[1,2]
e_c	[0,2]

Charts

A chart contains many Edges which record information regarding bits of the tree (or potential bits of tree).

An edge has a Span, indicating which words in the sentence this bit of tree relates to.



$edge_n$	Span
e_a	[0,1]
e_b	[1,2]
e_c	[0,2]
e_d	[0,3]

An edge that represents a successful parse will span the entire sentence. It will also have some property to indicate that it represents a tree that is grounded in the root node.

Components of an edge

An edge usually records the following information:

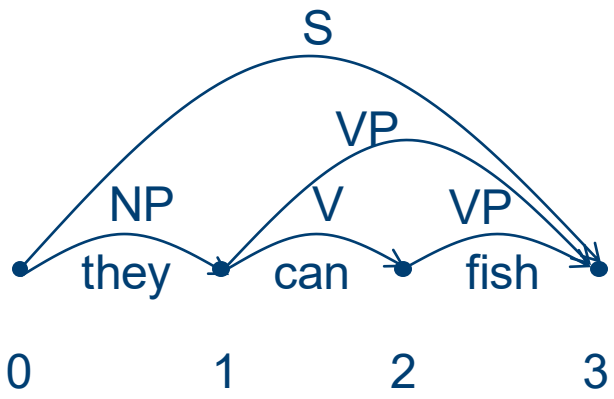
A unique identifier: this is a unique number which allows us to distinguish one edge from all the others in the chart.

A span: this is an indication of which words in the sentence this bit of tree relates to.

Functional information: Some information that categorises the tree piece represented in this edge.

A history: (optional) If this edge was created from previous edges then often this information is recorded. If we find a valid parse the history will tell us how we got there.

Components of an edge



$edge_n$	Functional Info	Span	History
⋮			
e_a	NP	[0,1]	
e_b	V	[1,2]	
e_c	VP	[2,3]	
e_d	VP	[1,3]	(e_b, e_c)
e_e	S	[0,3]	(e_a, e_d)

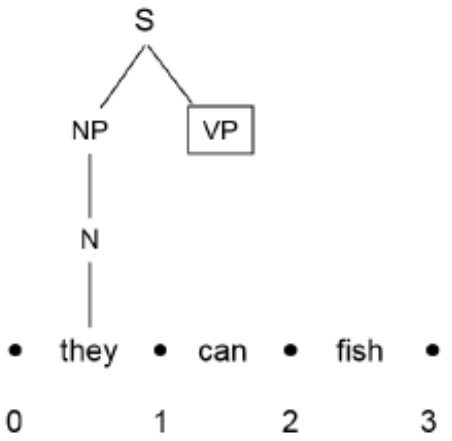
Summary

1. A toy CFG
2. Charts
3. **Earley parsing (Top-Down)**
4. Chart parsing (Bottom-Up, CKY Algorithm)
5. Parsing and Language Processing (Incremental Parsing)
6. Resolving Multiple Parses
7. Dependency Parsing

The Earley parser

The Earley parser is a top-down approach to searching the space of possible trees.

The functional information included in the Earley edges is called a Dotted Rule or Progress Rule.



$edge_n$	Dotted Rule	[starting at, waiting at]	History
⋮			
e_n	S → NP • VP	[0,1]	e_{n-1}

Dotted rules

For a grammar rule $A \rightarrow \alpha\beta$, we can write dotted forms as $A \rightarrow \bullet\alpha\beta \mid \alpha \bullet \beta \mid \alpha\beta\bullet$

$S \rightarrow NP\ VP\bullet$ ← A successful parse (assuming [0,N])

$S \rightarrow \bullet NP\ VP$

$edge_n$	Dotted Rule	[starting at, waiting at]	History
e_0	$S \rightarrow \bullet NP\ VP$	[0,0]	
⋮			
e_n	$S \rightarrow NP \bullet VP$	[0,1]	e_m
⋮			
e_z	$S \rightarrow NP \bullet VP$	[0,1]	e_y

A **dotted rule** is a production from a grammar annotated with a **dot (•)** that shows **how much of the right-hand side has been recognized** in the input.

Dotted rules

For a grammar rule $A \rightarrow \alpha\beta$, we can write dotted forms as $A \rightarrow \bullet\alpha\beta$ | $\alpha \bullet \beta$ | $\alpha\beta\bullet$

A **dotted rule** is a production from a grammar annotated with a **dot (•)** that shows **how much of the right-hand side has been recognized** in the input.

The dot moves **from left to right** as the parser recognizes input. It encodes both the **structure of the grammar** and the **current progress** in parsing.

Dot Position	**Meaning**	**Action for Parsing**
•αβ	Nothing Matched yet	PREDICT
α•β	α matched, β expected	Scan / Complete
αβ•	Entire RHS matched	Complete

Dotted rules

$S \rightarrow NP\ VP\bullet$

$S \rightarrow \bullet NP\ VP$ ← A unexplored S node that is looking for an NP

$edge_n$	Dotted Rule	[starting at, waiting at]	History
e_0	$S \rightarrow \bullet NP\ VP$	[0,0]	
⋮			
e_n	$S \rightarrow NP \bullet VP$	[0,1]	e_m
⋮			
e_z	$S \rightarrow NP \bullet VP$	[0,1]	e_y

Dotted rules

$S \rightarrow NP\ VP\bullet$

$S \rightarrow \bullet NP\ VP$

$edge_n$	Dotted Rule	[starting at, waiting at]	History
e_0	$S \rightarrow \bullet NP\ VP$	[0,0]	
⋮			
e_n	$S \rightarrow NP \bullet VP$	[0,1]	e_m
⋮			
e_z	$S \rightarrow NP \bullet VP$	[0,1]	e_y

An S node that has successfully parse an NP and is looking for a VP

Initialization

$edge_n$	Dotted Rule	[starting at, waiting at]	History
e_0	S → •NP VP	[0,0]	

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

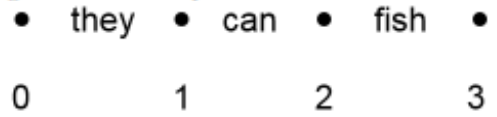


Figure 8: Remember that the input sentence we are trying to parse is *they can fish*.

Parser operators - Predictor

$edge_n$	Dotted Rule	[starting at, waiting at]	History
e_0	S → •NP VP	[0,0]	
e_1	NP → •N	[0,0]	
e_2	NP → •N PP	[0,0]	

predictor

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

• they • can • fish •

0 1 2 3

Figure 8: Remember that the input sentence we are trying to parse is *they can fish*.

Parser operators - Scanner

$edge_n$	Dotted Rule	[starting at, waiting at]	History
e_0	S → •NP VP	[0,0]	
e_1	NP → •N	[0,0]	
e_2	NP → •N PP	[0,0]	
e_3	N → they •	[0,1]	

scanner
N + they

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

• they • can • fish •

0 1 2 3

Figure 8: Remember that the input sentence we are trying to parse is *they can fish*.

Parser operators - Completer

$edge_n$	Dotted Rule	[starting at, waiting at]	History
e_0	S → •NP VP	[0,0]	
e_1	NP → •N	[0,0]	
e_2	NP → •N PP	[0,0]	
e_3	N → they •	[0,1]	
e_4	NP → N •	[0,1]	e_3
e_5	NP → N • PP	[0,1]	e_3
e_6	S → NP • VP	[0,1]	e_4

completer
N → they • [0,0]
NP→• N [0,0]

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

• they • can • fish •

0 1 2 3

Figure 8: Remember that the input sentence we are trying to parse is *they can fish*.

Earley parsing

$edge_n$	Dotted Rule	[starting at, waiting at]	History	word n
e_0	S → •NP VP	[0,0]		word 0
e_1	NP → •N	[0,0]		word 1
e_2	NP → •N PP	[0,0]		
e_3	N → they •	[0,1]		
e_4	NP → N •	[0,1]	e_3	
e_5	NP → N • PP	[0,1]	e_3	
e_6	S → NP • VP	[0,1]	e_4	
e_7	PP → •P NP	[1,1]		word 2
e_8	VP → •V	[1,1]		
e_9	VP → •V NP	[1,1]		
e_{10}	VP → •V VP	[1,1]		
e_{11}	VP → •V PP	[1,1]		
e_{12}	V → can •	[1,2]		
e_{13}	VP → V •	[1,2]	e_{12}	
e_{14}	VP → V • NP	[1,2]	e_{12}	
e_{15}	VP → V • VP	[1,2]	e_{12}	
e_{16}	S → NP VP •	[0,2]	e_4, e_{13}	
e_{17}	VP → VP • PP	[1,2]	e_{13}	
e_{18}	NP → •N	[2,2]		word 3
e_{19}	NP → •N PP	[2,2]		
e_{20}	VP → •V	[2,2]		
e_{21}	VP → •V NP	[2,2]		
e_{22}	VP → •V VP	[2,2]		
e_{23}	VP → •V PP	[2,2]		
e_{24}	PP → •P NP	[2,2]		
e_{25}	N → fish •	[2,3]		
e_{26}	V → fish •	[2,3]		
e_{27}	NP → N •	[2,3]	e_{25}	
e_{28}	NP → N • PP	[2,3]	e_{25}	
e_{29}	VP → V •	[2,3]	e_{26}	

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

• they • can • fish •

0 1 2 3

Figure 8: Remember that the input sentence we are trying to parse is *they can fish*.

Summary

1. A toy CFG
2. Charts
3. Earley parsing (Top-Down)
4. **Chart parsing (Bottom-Up, CKY Algorithm)**
5. Parsing and Language Processing (Incremental Parsing)
6. Resolving Multiple Parses
7. Dependency Parsing

CKY Parsing

We can also have a bottom-up approach to parsing. CKY (or CYK) parsing [**Cocke–Younger–Kasami algorithm**] is a classical bottom-up dynamic programming algorithm.

The algorithm proceeds by introducing each word from left to right (L-R parsing).

The Fundamental Rule of CKY/Chart Parsing

- If you have:
  - A partial edge that **expects `B` next**, and
  - A **completed edge for `B`** starting where the partial edge left off,
  - then you can **combine them** to advance the dot in the first edge

$$[A \rightarrow \alpha B \cdot \beta]\ [i : k]$$

If a partial edge `[A → α • B β] [i:j]` is expecting `B`, and a completed edge `[B → γ •] [j:k]` exists, then add the edge `[A → α B • β] [i:k]`.

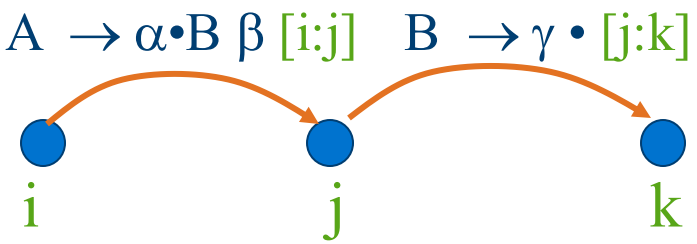
This allows CKY/CYK to combine smaller parses into larger parses.

The Fundamental Rule

This rule is applied in every chart parser.

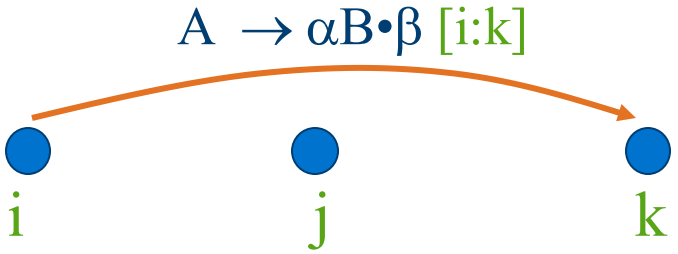
If the chart contains the edges:

- [A $\rightarrow \alpha \bullet B\ \beta$] [i:j]
- [B $\rightarrow \gamma \bullet$] [j:k]



then add the edge:

- [A $\rightarrow \alpha B \bullet \beta$] [i:k]



The CKY Algorithm

- CKY (Cocke–Kasami–Younger) algorithm requires **Chomsky Normal Form (CNF)** because of its reliance on **binary branching of grammar rules**.
  - A context-free grammar is in **CNF** if all its productions are of the form:
    - **Binary rules: A --> BC**
    - **Terminal Rules: A --> a**
  - There are no rules with more than 2 non-terminals on the right-hand side and no ε-productions (except possibly for the start symbol).

The CKY Algorithm

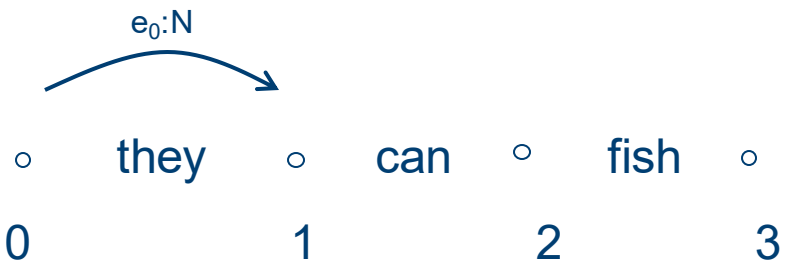
- The CKY algorithm is **bottom-up** and builds a **parse chart** in a table where each cell `[i,j]` represents the substring from word `i` to word `j`.
- CKY works by **splitting substrings into two parts** recursively and combining them using grammar rules.
- If every rule is **binary**, you can systematically combine two smaller constituents into a larger one.
- CKY looks for `B` spanning `[i,k]` and `C` spanning `[k,j]` and then adds `A` spanning `[i,j]`.
- This **binary structure** is essential because CKY's dynamic programming table only handles **pairs of adjacent spans** efficiently.

The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.



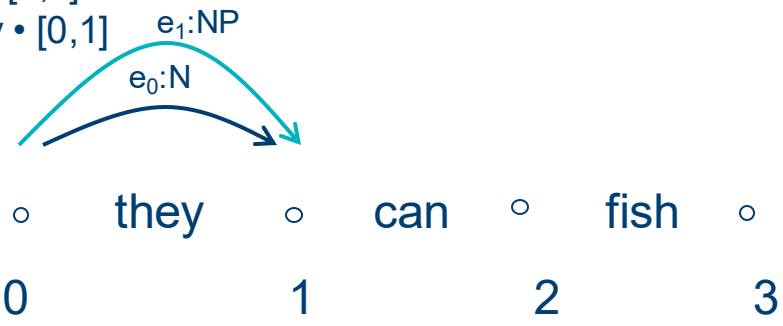
The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0

$S \rightarrow NP\ VP$
$NP \rightarrow N\ PP$
$NP \rightarrow N$
$PP \rightarrow P\ NP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ VP$
$VP \rightarrow V\ NP$
$VP \rightarrow V$
$N \rightarrow$ {it,fish,rivers, ...}
$P \rightarrow$ {in}
$V \rightarrow$ {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

NP→ • N [0,0]
N → they • [0,1]

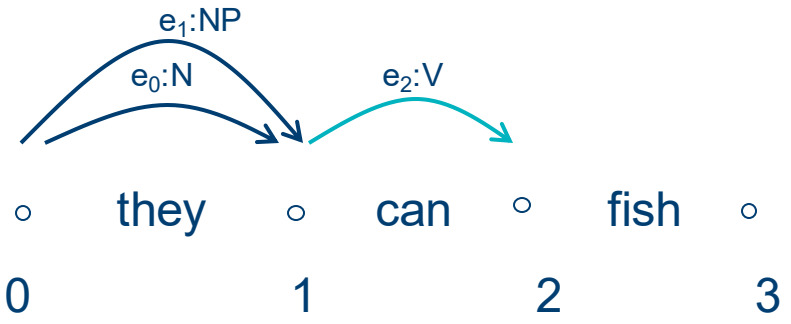


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

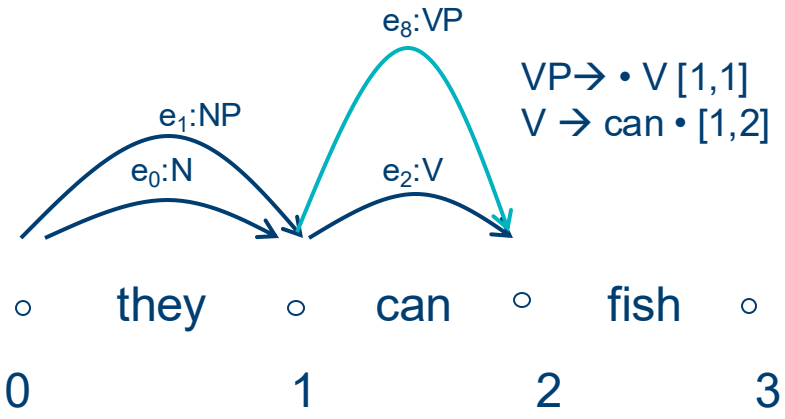


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*
e_3	VP	[1,2]	e_2

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

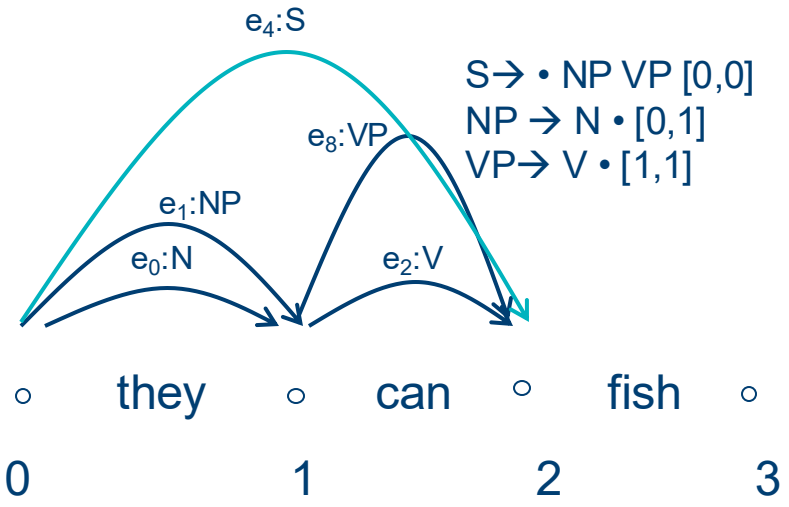


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*
e_3	VP	[1,2]	e_2
e_4	S	[0,2]	e_1, e_3

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

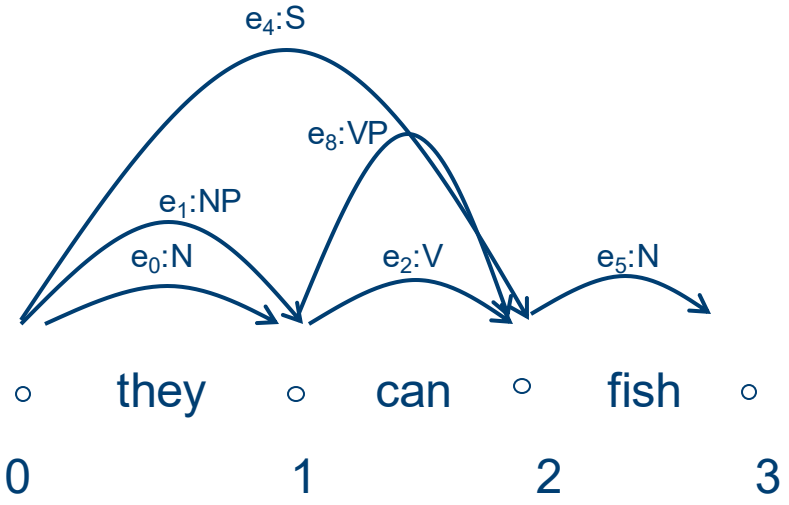


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*
e_3	VP	[1,2]	e_2
e_4	S	[0,2]	e_1, e_3
e_5	N	[2,3]	*fish*

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

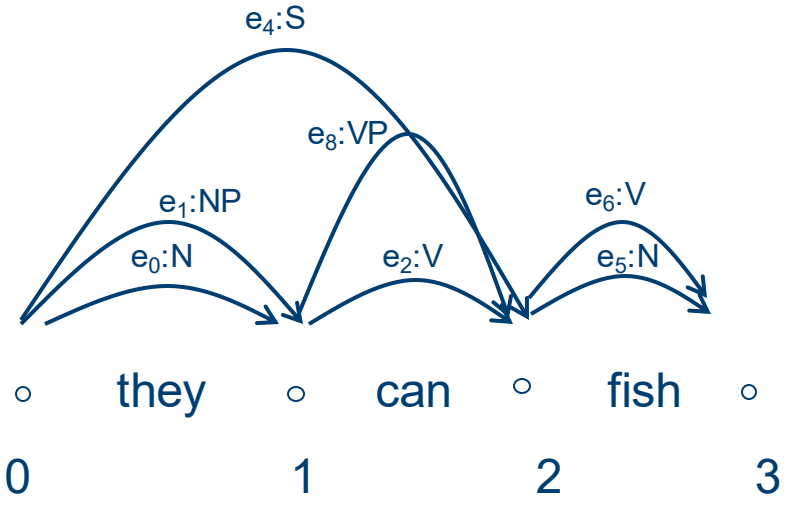


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*
e_3	VP	[1,2]	e_2
e_4	S	[0,2]	e_1, e_3
e_5	N	[2,3]	*fish*
e_6	V	[2,3]	*fish*

$S \rightarrow NP\ VP$
$NP \rightarrow N\ PP$
$NP \rightarrow N$
$PP \rightarrow P\ NP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ VP$
$VP \rightarrow V\ NP$
$VP \rightarrow V$
$N \rightarrow$ {it,fish,rivers, ...}
$P \rightarrow$ {in}
$V \rightarrow$ {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

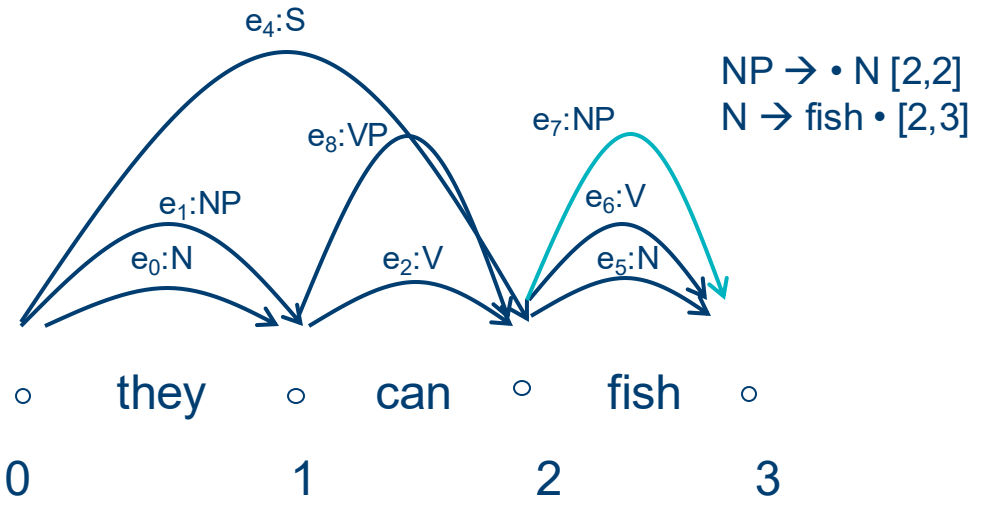


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*
e_3	VP	[1,2]	e_2
e_4	S	[0,2]	e_1, e_3
e_5	N	[2,3]	*fish*
e_6	V	[2,3]	*fish*
e_7	NP	[2,3]	e_5

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

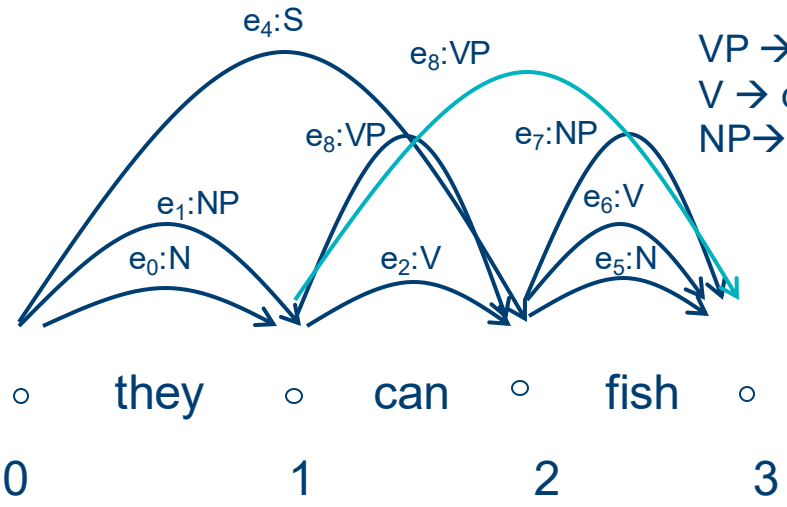


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*
e_3	VP	[1,2]	e_2
e_4	S	[0,2]	e_1, e_3
e_5	N	[2,3]	*fish*
e_6	V	[2,3]	*fish*
e_7	NP	[2,3]	e_5
e_8	VP	[1,3]	e_2, e_7

S → NP VP
NP → N PP
NP → N
PP → P NP
VP → VP PP
VP → V VP
VP → V NP
VP → V
N → {it,fish,rivers, ...}
P → {in}
V → {can,fish}

Figure 7: Here are the production rules again for our toy CFG.

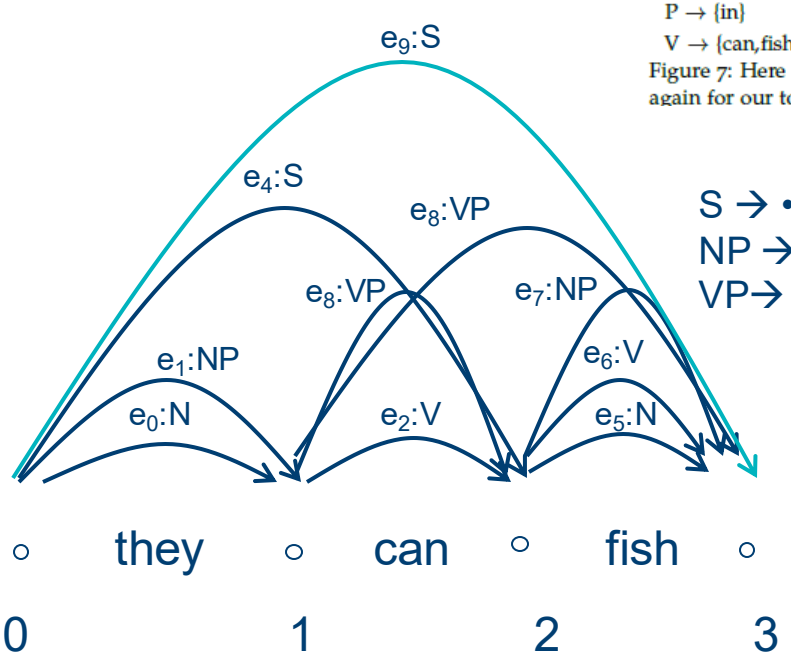


The Chart parser

$edge_n$	Category	Span	History
e_0	N	[0,1]	*they*
e_1	NP	[0.1]	e_0
e_2	V	[1,2]	*can*
e_3	VP	[1,2]	e_2
e_4	S	[0,2]	e_1, e_3
e_5	N	[2,3]	*fish*
e_6	V	[2,3]	*fish*
e_7	NP	[2,3]	e_5
e_8	VP	[1,3]	e_2, e_7
e_9	S	[0,3]	e_1, e_8

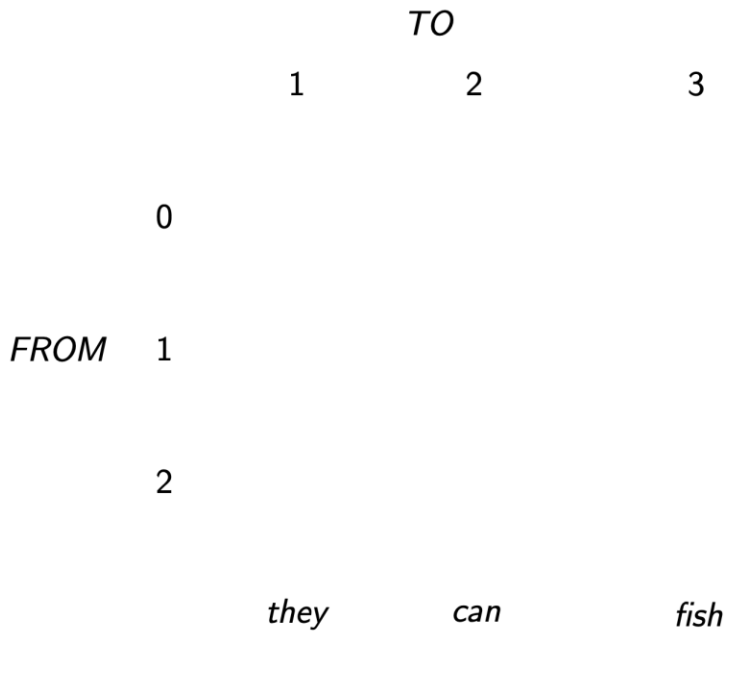
$S \rightarrow NP\ VP$
$NP \rightarrow N\ PP$
$NP \rightarrow N$
$PP \rightarrow P\ NP$
$VP \rightarrow VP\ PP$
$VP \rightarrow V\ VP$
$VP \rightarrow V\ NP$
$VP \rightarrow V$
$N \rightarrow$ {it,fish,rivers, ...}
$P \rightarrow$ {in}
$V \rightarrow$ {can,fish}

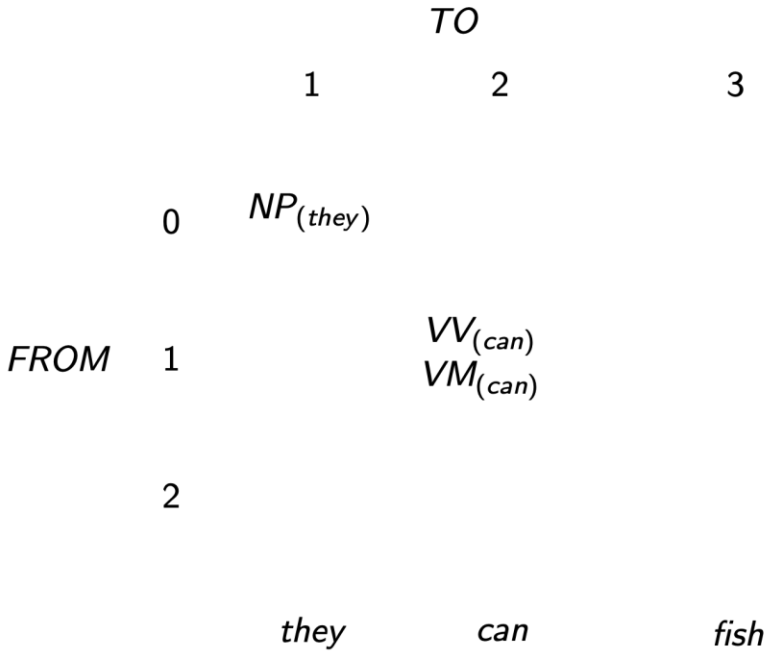
Figure 7: Here are the production rules again for our toy CFG.



S → • NP VP [0,0]
NP → N • [0,1]
VP→ V NP • [1,3]

The CKY/CYK Chart Derivation





The CKY/CYK Chart Derivation (2)

		TO 1	2	3
	0	$NP_{(they)}$	.	
FROM	1		$VV_{(can)}$ $VM_{(can)}$	
	2			$VV_{(fish)}$ $NP_{(fish)}$
		they	*can*	*fish*

		TO 1	2	3
	0	$NP_{(they)}$	.	
FROM	1		$VV_{(can)}$ $VM_{(can)}$	$VP_{1\rightarrow([1,2]_{VV},[2,3]_{NP})}$ $VP_{2\rightarrow([1,2]_{VM},[2,3]_{VV})}$
	2			$VV_{(fish)}$ $NP_{(fish)}$
		they	*can*	*fish*

The CKY/CYK Chart Derivation (3)

		TO 1	2	3
FROM	0	$NP_{(they)}$	.	$S_{1\rightarrow([0,1]_{NP},[1,3]_{VP_1})}$ $S_{2\rightarrow([0,1]_{NP},[1,3]_{VP_2})}$
	1		$VV_{(can)}$ $VM_{(can)}$	$VP_{1\rightarrow([1,2]_{VV},[2,3]_{NP})}$ $VP_{2\rightarrow([1,2]_{VM},[2,3]_{VV})}$
	2			$VV_{(fish)}$ $NP_{(fish)}$
		they	*can*	*fish*

Beyond CKY: Span-Based Neural Constituency Parsing (Brief Mention)

CKY doesn't disambiguate among the possible parses. We need to solve this **disambiguation problem.**

One recent neural parsing approach called **span-based constituency parsing (neural CKY, Kitaev et al 2019).**

Choosing a single parse from all possible parses (disambiguation) can be done by neural constituency parsers.

Span-based neural constituency parses train a neural classifier to assign a score to each constituent, and then use a modified version of CKY to combine these constituent scores to find the best-scoring parse tree.

(See SLP3, Ch18 for further detail: https://web.stanford.edu/~jurafsky/slp3/18.pdf)

Summary

1. A toy CFG
2. Charts
3. Earley parsing (Top-Down)
4. Chart parsing (Bottom-Up, CKY Algorithm)
5. **Parsing and Language Processing (Incremental Parsing)**
6. Resolving Multiple Parses
7. Dependency Parsing

Parsing and Language Processing

- **Chart parsing leaves most of the computational cost until the end, which is not a good match for cognitive processing.**

Consider the following sentences:
(i) John saw the dog. (ii) Dogs John saw smiled.

Assume a psycholinguistic experiment showed that by the second word (ii) is harder to process than (i).

Would we want a parsing algorithm to make a similar prediction?

Parsing and Language Processing

- **Chart parsing leaves most of the computational cost until the end, which is not a good match for cognitive processing.**

Consider the following sentences:
(i) John saw the dog. (ii) Dogs John saw smiled.

Assume a psycholinguistic experiment showed that by the second word (ii) is harder to process than (i).

Would we want a parsing algorithm to make a similar prediction?

This early cost would not be predicted by purely bottom-up parsers (e.g., standard chart parsing).

Parsing and Language Processing

- **Chart parsing leaves most of the computational cost until the end, which is not a good match for cognitive processing.**

Consider the following sentences:
(i) John saw the dog. (ii) Dogs John saw smiled.

Assume a psycholinguistic experiment showed that by the second word (ii) is harder to process than (i).

Would we want a parsing algorithm to make a similar prediction?

This early cost would not be predicted by purely bottom-up parsers (e.g., standard chart parsing).

BUT there are some parsing algorithms that model incremental expectations

Parsing and Language Processing

- **Chart parsing leaves most of the computational cost until the end, which is not a good match for cognitive processing.**

Consider the following sentences:
(i) John saw the dog. (ii) Dogs John saw smiled.

Assume a psycholinguistic experiment showed that by the second word (ii) is harder to process than (i).

Would we want a parsing algorithm to make a similar prediction?

This early cost would not be predicted by purely bottom-up parsers (e.g., standard chart parsing).

BUT there are some parsing algorithms that model incremental expectations

Parsing and Language Processing

- Human parsing is fast, good-enough and incremental. Parsing problems occur if more than one structure is compatible with the input either at (1) the current point but not later (local ambiguity) or (2) for the input overall (global ambiguity).
- Consider the garden path sentence:
  - *the old man the boats*.
- A (serial/limited parallel— i.e. one where the correct structure is not considered initially) bottom-up parser doesn't realize a mistake until it reaches the end of the sentence and cannot create a full parse.
- CKY parsing cannot model humans – it is too parallel, or (if limited), also doesn't recognize garden paths immediately.

A *very brief* mention of Incremental Parsing Algorithms (Computational Psycholinguistics)

- **John Hale (2001, 2003):** a **probabilistic Earley parser** is used as a model of online eager sentence processing. It is a model of **total parallelism**.
- It assumes cognitive effort expended to parse a given prefix is proportional to the total probability of all the structural analyses which are not compatible with the prefix.

Find out more!

John Hale. 2001. A probabilistic Earley parser as a psycholinguistic model. In Second Meeting of the North American Chapter of the Association for Computational Linguistics.

A *very brief* mention of other Computational Psycholinguistics approaches

- Yngve (1960): size of stack (containing all nodes left to explore) correlates with **working memory load.**

This is the malt that the rat that the cat that the dog worried killed ate.

STACK: N VP VP VP

This is the malt that was eaten by the rat that was killed by the cat that was worried by the dog.

Example from Paula Buttery's Formal Models of Language (Lecture 3)

Summary

1. A toy CFG
2. Charts
3. Earley parsing (Top-Down)
4. Chart parsing (Bottom-Up, CKY Algorithm)
5. Parsing and Language Processing (Incremental Parsing)
6. **Resolving Multiple Parses**
7. Dependency Parsing

Resolving multiple parses

When parsing with larger and more representative grammars there are often a very large number of parses for each sentence.

How do we select the most probable parse?

- Use part-of-speech tagging to reduce the structural ambiguity caused by POS ambiguity.
- Use a probabilistic context free grammar (PCFG) and statistical parsing.

Our toy CFG

S	→	NP VP
NP	→	Pron
NP	→	Det N
NP	→	N
NP	→	NP PP
VP	→	V
VP	→	V NP
VP	→	VP PP
PP	→	P NP
Det	→	{a, the, this, these}
N	→	{river, rivers, fish, December}
Pron	→	{they, them, she, her, he, him}
V	→	{eat, eats, fish, fishes}
P	→	{in, on}

Chart parsing

- How do we determine if a sentence is licensed by a grammar?
- Remember a CFG is a deliberately constrained model!
- Recall *dynamic programming* that we saw for minimum edit distance and for part-of-speech tagging
    - Iteratively solve a problem by reducing it to a smaller problem
    - In this case: consider parsing shorter sequences of words
    - This is more complex than working with FSAs

Parse Ranking with a Probabilistic Context-Free Grammar

S → NP VP [0.8]

NP → Det N [0.7]

VP → V [1.0]

Det → 'The' [0.9]

N → 'cat' [0.6]

V → 'sleeps' [0.5]

Parse Ranking with a Probabilistic Context-Free Grammar

- We need these components:
  - Data: input x (text), output y (parse tree)
  - Model: $P(y|x) \propto \Pi_i P(\text{rule}|y_i)$ [imprecise but hopefully intuitive notation]
  - Objective: P(x,y)
  - Training: count! (Maximum Likelihood Estimate)
  - Test: chart parsing plus ranking by probability

Key idea: the probability of the whole tree is built up from probabilities of each rule used in the derivation.

Parse Ranking with a Probabilistic Context-Free Grammar

- Each node expansion contributes **one conditional probability**.
- The total probability of the tree is the **product of these probabilities**.
- During **training**, we count how often each rule is used and normalize to estimate probabilities.
- At **test time**, we generate all possible trees with chart parsing and pick the one with the highest probability.

$$P(y \mid x) \propto P(S \rightarrow NP\ VP) \times P(NP \rightarrow Det\ N) \times P(VP \rightarrow V) \times P(Det \rightarrow' The') \times P(N \rightarrow' cat') \times P(V \rightarrow' sleeps')$$

Parse Ranking with Logistic Regression (or “MaxEnt”)

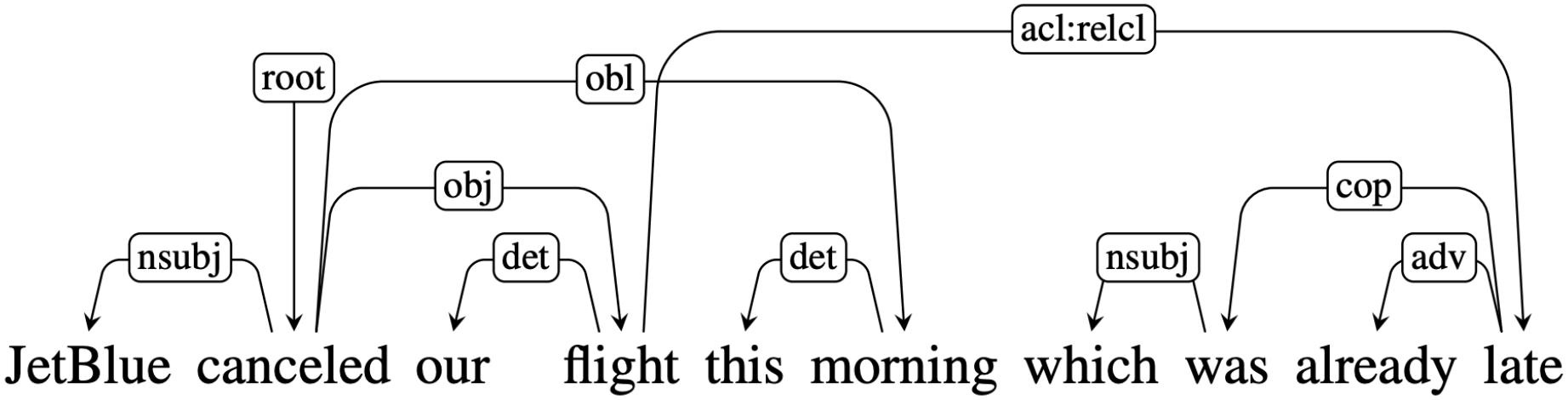
- We need these components:
  - Data: input x (text), output y (parse tree)
  - Model: $P(y|x) \propto \exp (\Sigma_i f_i(x,y))$ [imprecise but hopefully intuitive notation]
  - Objective: $P(y|x)$
  - Training: gradient descent
  - Test: chart parsing plus ranking by probability

Summary

1. A toy CFG
2. Charts
3. Earley parsing (Top-Down)
4. Chart parsing (Bottom-Up, CKY Algorithm)
5. Parsing and Language Processing (Incremental Parsing)
6. Resolving Multiple Parses
7. **Dependency Parsing**

Dependency Parsing

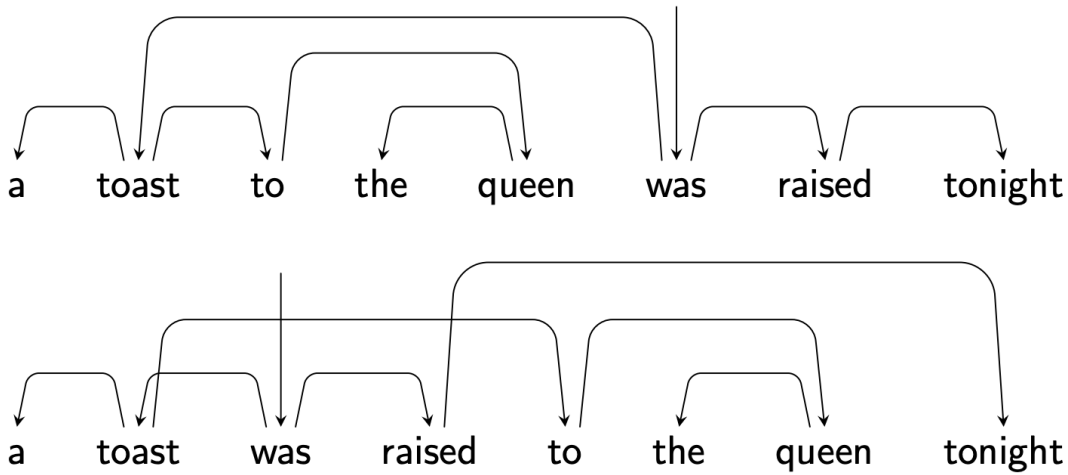
- A dependency tree is a directed graph representation of a string—each edge represents a grammatical relationship between the symbols. We consider **head** and **dependent** relations.



Example from SLP3 Ch19, https://web.stanford.edu/~jurafsky/slp3/19.pdf

Valid dependency trees may be projective or non-projective

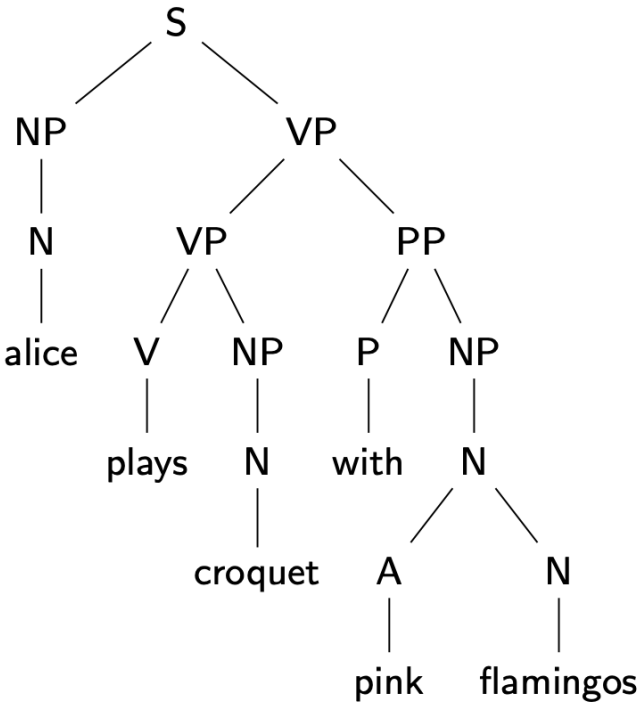
- Dependency trees can be described as being projective or nonprojective—loosely speaking, a projective tree can be drawn without edges crossing whereas a non-projective tree cannot.
- The difference between projective and non-projective dependencies can have implications for parsing complexity.

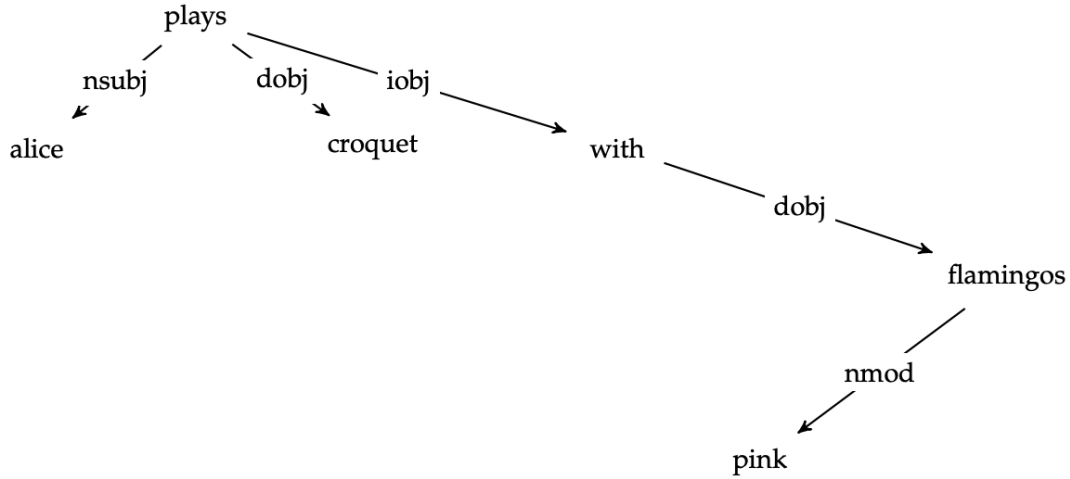


Valid dependency trees may be projective or non-projective

- For the most part, the dependents of a given head form a contiguous substring of the sentence, i.e., all the nodes occurring between the head and its dependent are (transitively) dominated by the head. Such dependencies have been termed projective
- In addition to projective dependencies, we also find instances where the dependents of a head are discontinuous. This happens when a node in the span of a head and its dependents is not (directly or indirectly) dominated by the head. Such dependencies are known as crossing or non-projective dependencies.

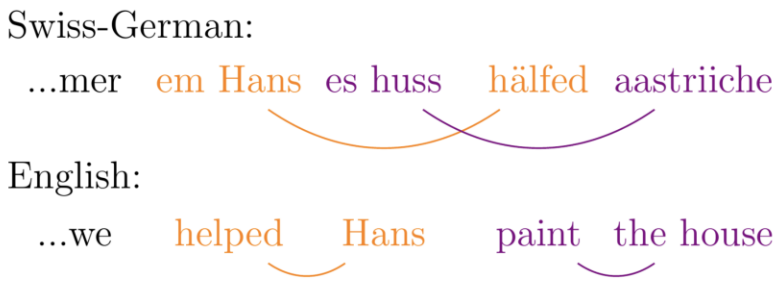
There is a *weak equivalence* between Projective Dependency and Constituency Parsing





Crossing or Non-Projective Dependencies: Swiss German

- From last week, crossing dependencies indicate deviations from context-free grammar (Marcus, 1965; Shieber, 1985).
  - More specifically, the hierarchy of mildly context-sensitive languages is defined by restrictions on gap degree. There is a 'limited amount of cross-serial dependencies' allowed in TAG derivations (Joshi, 1985).
- But, crossing dependencies are rare! From a cognitive modelling perspective, there have been attempts to explain this in dependency-based perspectives including **Minimum Dependency Length (MDL)**.

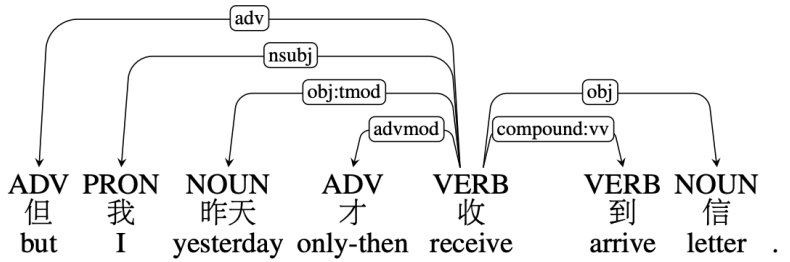


Universal Dependencies

- The largest open community project for building dependency trees is the Universal Dependencies project at https://universaldependencies.org/, which currently has almost 200 dependency treebanks in more than 100 languages (de Marneffe et al., 2021).
- Important for low-resourced NLP; dependency grammars are a flexible formalism for linguistic annotation, useful for a typologically-consistent schema.

de Marneffe, M.-C., C. D. Manning, J. Nivre, and D. Zeman. 2021. Universal Dependencies. Computational Linguistics, 47(2):255–308.

Open class words	Closed class words	Other
ADJ	ADP	PUNCT
ADV	AUX	SYM
INTJ	CCONJ	X
NOUN	DET	
PROPN	NUM	
VERB	PART	
	PRON	
	SCONJ	



[Chinese] 但我昨天才收到信 "But I didn't receive the letter until yesterday

Similar Parsing Algorithms exist for Dependency Parsing

- We will not be introducing dependency parsing algorithms formally, but here are the main ideas behind classical dependency parsing algorithms (see SLP3 Ch19, https://web.stanford.edu/~jurafsky/slp3/19.pdf).
- **Transition-based Dependency Parsing:** a stack on which we build the parse, a buffer of tokens to be parsed, and a parser which takes actions on the parse via a predictor called an oracle.
- More recently, **graph-based dependency parsers (e.g., Dozat & Manning 2017)** introduce state-of-the-art graph-based multilingual parsers are based on neural networks.
  - These algorithms score entire trees and unlike transition-based approaches, graph-based parsers can produce non-projective trees
  - Algorithms (1) assigning a score to each edge, and (2) finding the best parse tree given the scores of all potential edges.

Summary of Key Points

- Classical Constituency Parsing Algorithms using Dynamic Programming: Top-Down Earley Parser, Bottom-Up CKY/CYK Algorithm for Chart Parsing
- Incremental Parsing, and connections to computational psycholinguistics.
- Probabilistic Context Free Grammars (PCFGs) and parse reranking (e.g., in ambiguity)
- Dependency Grammar Formalism, Universal Dependencies and a brief outline of Dependency Parsing algorithms.

Further Reading

- J&M (2nd edition), Chapters 13 and 14
- SLP3 Ch18 and Ch19
- John Hale. 2001. A probabilistic Earley parser as a psycholinguistic model. In Second Meeting of the North American Chapter of the Association for Computational Linguistics.
- Kitaev, N., S. Cao, and D. Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. ACL. https://aclanthology.org/P19-1340/.
- Temperley, D., & Gildea, D. (2018). Minimizing syntactic dependency lengths: Typological/cognitive universal?. *Annual Review of Linguistics*, *4*(1), 67-80.
- **Universal Dependencies:**
  - A paper critiquing UD: Osborne, T. & Gerdes, K., (2019) "The status of function words in dependency grammar: A critique of Universal Dependencies (UD)", *Glossa: a journal of general linguistics* 4(1): 17. doi: https://doi.org/10.5334/gjgl.537

Exercises (supervision 3, part 2/2)

- Using the toy grammar, apply chart parsing to the sentence *they fish in rivers in December*. Draw constituency trees corresponding to the complete parses of the sentence.
- How would a PCFG rank the parses of the above sentence? What additional information might be useful for ranking (which a PCFG does not consider)?
- Draw dependency parses corresponding to the constituency parses of the above sentence.

Pre-Lecture Exercises for Lent Term

How can we make sense of a model trained on 1,000 human lifetimes of language data?

When applying a model in a real-world setting, what ethical aspects need to be considered?