Grammars and Parsing

Alberto Lavelli
FBK-irst
lavelli@fbk.eu

1. **Context Free Grammars (CFGs)**
2. Efficiency and Expressivity
3. Features and Unification
4. Dependency Grammars
5. Resolving Ambiguity
6. Treebanks and Evaluation

1. Context Free Grammars

- Grammars
- Context Free Grammars (CFGs)
- Basic Parsing Strategies for CFGs
  - Top-Down
  - Bottom-Up
- Parsing and Search
- Redundancy in Parsing

Grammars

- A *grammar* is a 4-tuple $G = (N, \Sigma, P, S)$, where
  - N is a finite set of *nonterminal symbols*
  - Σ is a finite set of *terminal symbols*, disjoint from N
  - P is a set of rules, i.e. a finite subset of

    $$(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

    Productions $(\alpha, \beta) \in P$ are usually written $\alpha \rightarrow \beta$
  - S is a distinguished symbol in N called the start symbol

Chomsky hierarchy

Different types of grammars/languages according to the definition of *P*:

- Regular grammars/languages
- Context-Free grammars/languages
- Context-Sensitive grammars/languages
- Unrestricted grammars/languages

Rules

- Regular Grammars:

$$A \rightarrow xB$$

$$A \rightarrow x$$

where A and B are in $\boldsymbol{N}$ and x is in $\boldsymbol{\Sigma}$*

- Context-Free Grammars:

$$A \rightarrow \alpha$$

where A is in $\boldsymbol{N}$ and α is in $(\boldsymbol{N} \cup \boldsymbol{\Sigma})^*$

- Context-Sensitive Grammars:

$$\alpha \rightarrow \beta$$

where $|\alpha| \leq |\beta|$

Phrase Structure

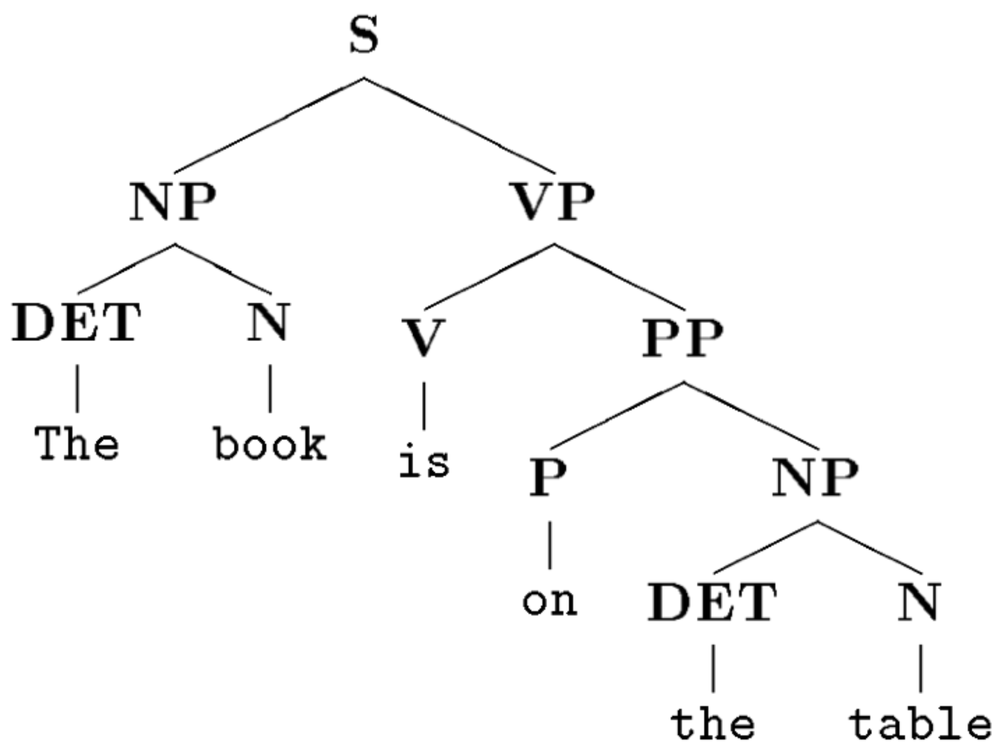
- *Language* = collection of *strings*
  but …
- Importance of hierarchical structure as well as linear structure of a given sentence

  the book is on the table

Sentence:

the book is on the table

Parse tree:



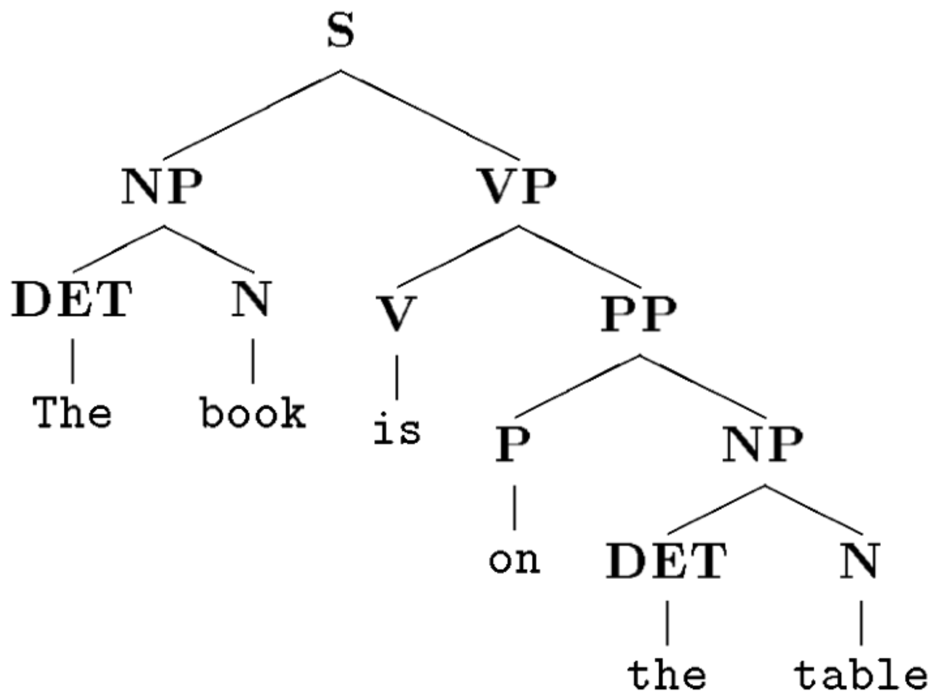
- Lexical elements:
  - *the* (DET)
  - *book, table* (Noun)
  - *is* (Verb)
  - *on* (Preposition)
- Constituent phrases:
  - *the book* (Noun Phrase)
  - *the table* (Noun Phrase)
  - *on the table* (Prepositional Phrase)
  - …

Phrase Structure

Constituents can be indicated either by bracketing

[$_{S}$ [$_{NP}$ [$_{DET}$ *the*] [$_{N}$ *book*]] [$_{VP}$ [$_{V}$ *is*] [$_{PP}$ *on* [$_{NP}$ [$_{DET}$ *the*] [$_{N}$ *table*]]]]]

or by means of parse trees



Phrase Structure

- Hierarchical information about constituents (dominance)
- Linear precedence information
- Labelling information (syntactic categories)

Context-Free Grammars

Phrase structure grammars (PSGs) provide a means of characterizing the structure of sentences

A Context-Free (Phrase Structure) Grammar consists of a set of rules of the following form:

$$A \rightarrow X_1\ X_2\ \ldots\ X_k\ (k \geq 0)$$

- A is a *nonterminal* (a category name; e.g. N, NP, VP, DET, etc.)
- each X_i is either a *nonterminal* or a terminal (i.e. a word)

An example of a simple CFG

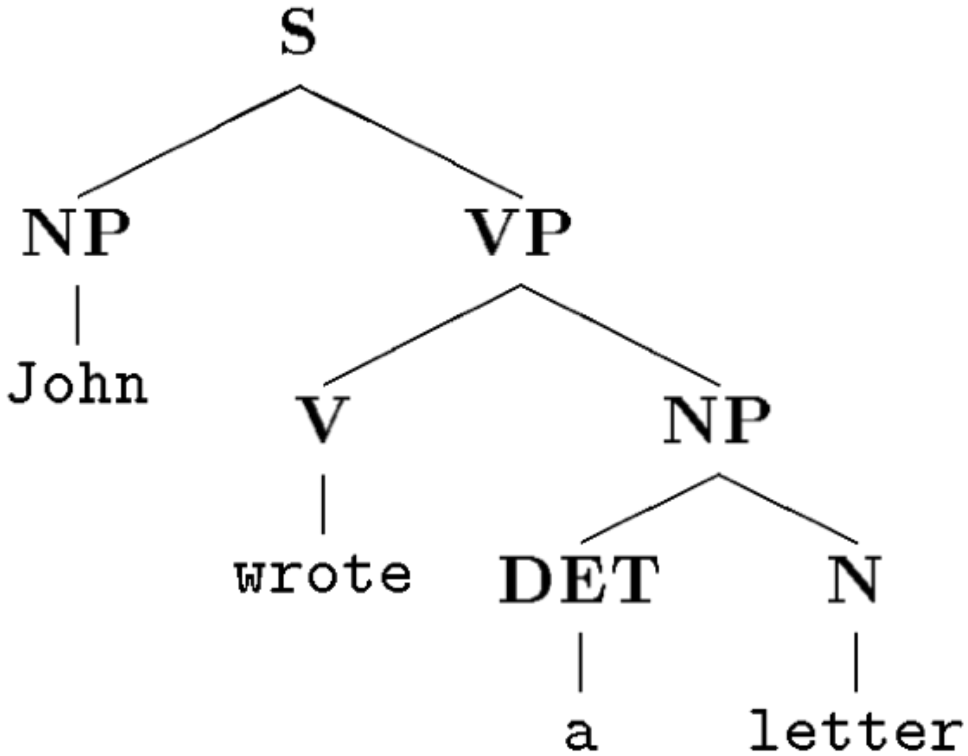
1. S → NP VP
2. NP → *John*
3. NP → *Mary*
4. NP → DET N
5. DET → *a*
6. N → *letter*
7. VP → V NP
8. VP → V NP PP
9. VP → V PP
10. V → *wrote*
11. PP → P NP
12. P → *to*

John wrote a letter

John wrote a letter to Mary

John wrote to Mary

…



Three questions

- Are there effective procedures for recognition/ generation of CFGs?
- How do we use CFGs to *parse* (i.e. assign structure to) strings?
- How do CFGs compare with FSLs computationally/descriptively?

Basic Parsing Strategies

Top-Down: A *goal-driven* strategy:

1. assume you are looking for *S* (i.e. sentence);
2. use rules ‘forward’ to ‘expand’ symbols until input is derived (else **fail**)

Bottom-Up: A *data-driven* strategy:

1. assume you are looking for *S*;
2. use rules ‘backward’ to ‘combine’ symbols until you get *S* (else **fail**)

Basic Parsing Strategies

Other dimensions:

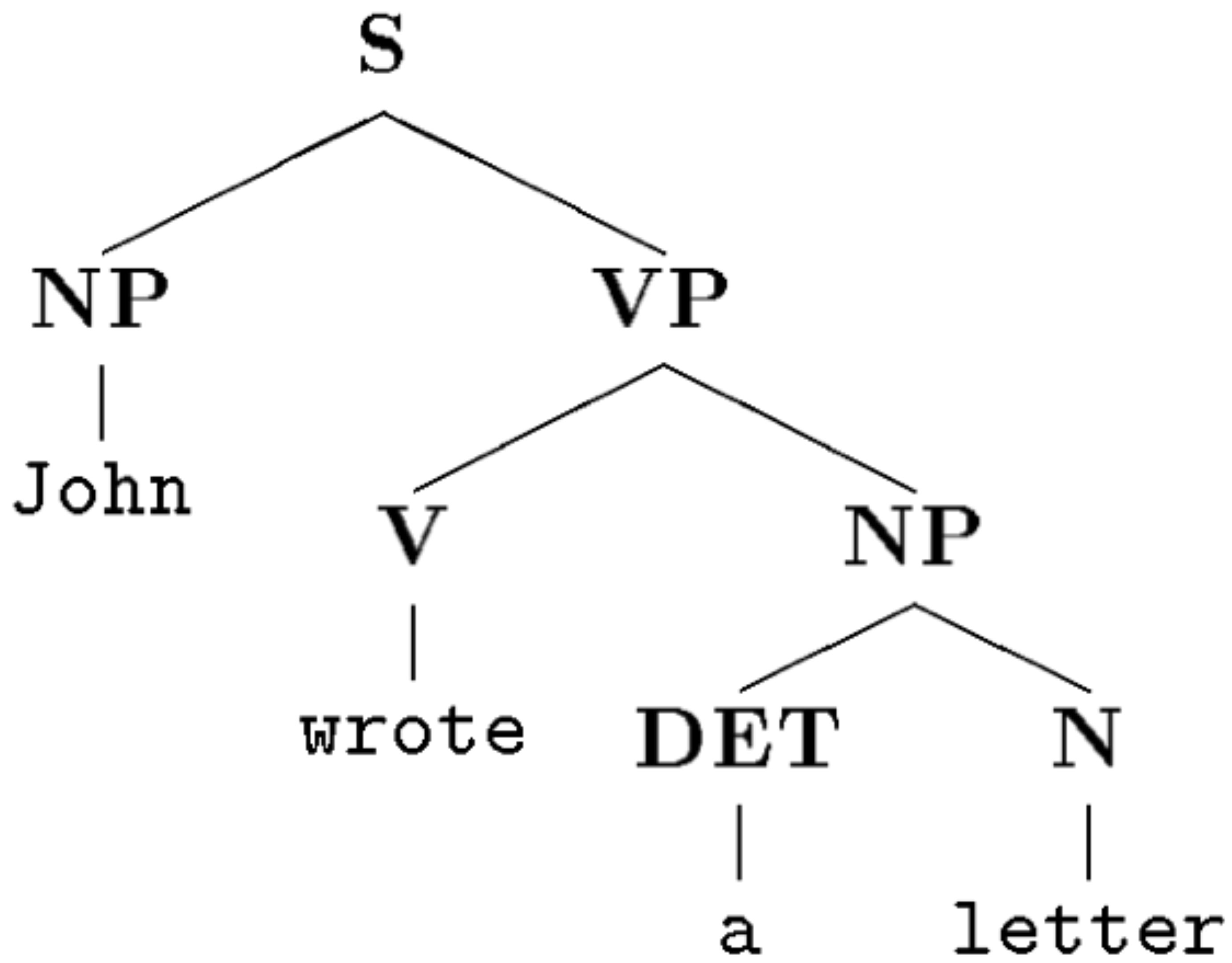
- *left-to-right* vs. *right-to-left* (but also *head-driven* or *island-driven*)
- *depth-first* vs. *breadth-first*

In the following examples, *left-to-right* and *depth-first* are usually adopted

Top-Down Strategy

Input: *John wrote a letter*

1	S	:	*John wrote a letter*	
2	NP VP	:	*John wrote a letter*	S → NP VP
3	VP	:	*wrote a letter*	NP → *John*
4	V NP	:	*wrote a letter*	VP → V NP
5	NP	:	*a letter*	V → *wrote*
6	DET N	:	*a letter*	NP → DET N
7	N	:	*letter*	DET → *a*
8		:		N → *letter*


S
NP
VP
John
V
NP
wrote
DET
N
a
letter

Crucial Points (1)

Non-determinism: at step 4, we could have chosen to 'expand' VP according to rule 8:

3	VP	:	*wrote a letter*	
4	V NP PP	:	*wrote a letter*	VP → V NP PP

Need some way of exploring the possibilities and recovering if necessary (backtracking)

Crucial Points (2)

Left recursion: a problem for top-down strategy.
If we added a new rule:

(13) VP → VP PP

3	VP	:	*wrote a letter*	
4	VP PP	:	*wrote a letter*	VP → VP PP
5	VP PP PP	:	*wrote a letter*	VP → VP PP

and so on…

Bottom-Up Strategy

Input: *John wrote a letter*

1	*John wrote a letter*	
2	NP *wrote a letter*	NP → *John*
3	NP V *a letter*	V → *wrote*
4	NP V DET *letter*	DET → *a*
5	NP V DET N	N → *letter*
6	NP V NP	NP → DET N
7	NP VP	VP → V NP
8	S	S → NP VP

Crucial Points

Empty productions: a problem for bottom-up strategy. Empty productions have the form:

$$A \rightarrow \varepsilon$$

E.g.:

NP → DET AP N
AP → ε
AP → ADJ AP
ADJ → *lengthy*
ADJ → *interesting*

Crucial Points

These new rules allow NPs such as:

a lengthy letter

a lengthy interesting letter

a letter

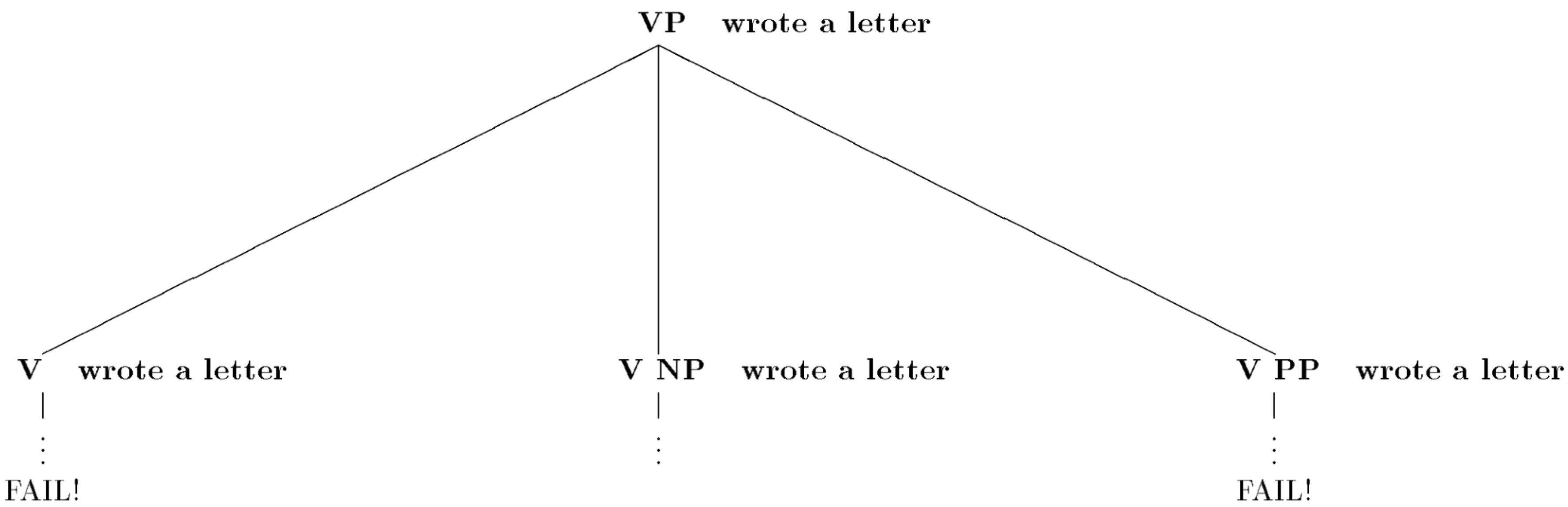
Note, however, that the rule AP $\rightarrow$ ε is always applicable!

Parsing and Search

In general, CFG parsing is *non-deterministic*.

Top-Down Example:

At different stages in the parsing process, more than one rule may be applicable:



Parsing and Search

Parsing algorithms need to explore the search space systematically.

To recover from errors, it is necessary to record the state of a parse each time a choice occurs.

Parsing and Search

E.g., considering the previous example,

The parse state:

VP : *wrote a letter*

has three different successor states:

V : *wrote a letter*

V NP : *wrote a letter*

V NP PP : *wrote a letter*

Parsing and Search

Parse maintains a list of parse states called an *agenda*:

- remove states from agenda;
- generate successor states;
- add successors to agenda;

Parse terminates successfully if the goal state (:) is generated.

Parse terminates unsuccessfully if it runs out of parse states to explore (i.e. the agenda is empty).

Parsing and Search

Search strategy:

this is determined by the order in which agenda items are considered:

Rule $S \rightarrow S_1\ S_2\ \ldots\ S_k$

σ = rest on agenda

Depth-first:

$$S\ \sigma \Rightarrow S_1\ S_2\ \ldots\ S_k\ \sigma$$

Breadth-first:

$$S\ \sigma \Rightarrow \sigma\ S_1\ S_2\ \ldots\ S_k$$

Redundancy in Parsing

Input: *John sang a song*

1	S	:	*John sang a song*	
2	NP VP	:	*John sang a song*	S → NP VP
3	VP	:	*sang a song*	NP → *John*
4	V NP PP	:	*sang a song*	VP → V NP PP
5	NP PP	:	*a song*	V → *sang*
6	DET N PP	:	*a song*	NP → DET N
7	N PP	:	*song*	DET → *a*
8	PP	:		N → *song*
9	P NP	:		PP → P NP

Redundancy in Parsing

4'	V NP	:	*sang a song*	VP → V NP
5'	NP	:	*a song*	V → *sang*
6'	DET N	:	*a song*	NP → DET N
7'	N	:	*song*	DET → *a*
8'		:		N → *song*

1. Context Free Grammars (CFGs)
2. **Efficiency and Expressivity**
3. Features and Unification
4. Dependency Grammars
5. Resolving Ambiguity
6. Treebanks and Evaluation

2. Efficiency and Expressivity

- Efficiency
  - Redundancy in Parsing
  - Active Chart Parsing
- Expressivity
  - comparing CFGs and FSAs
  - Pros and Cons of CFGs
    - Agreement, subcategorization, …

Chart Parsing

Dynamic programming technique which keeps track of what has been done and of partial hypotheses. Resulting data structure is called the *active chart*.

The chart contains data structures called *edges*, which represent (partially) recognized constituents.

Dotted Rules

'Dotted Rules': edges have labels of the general form:

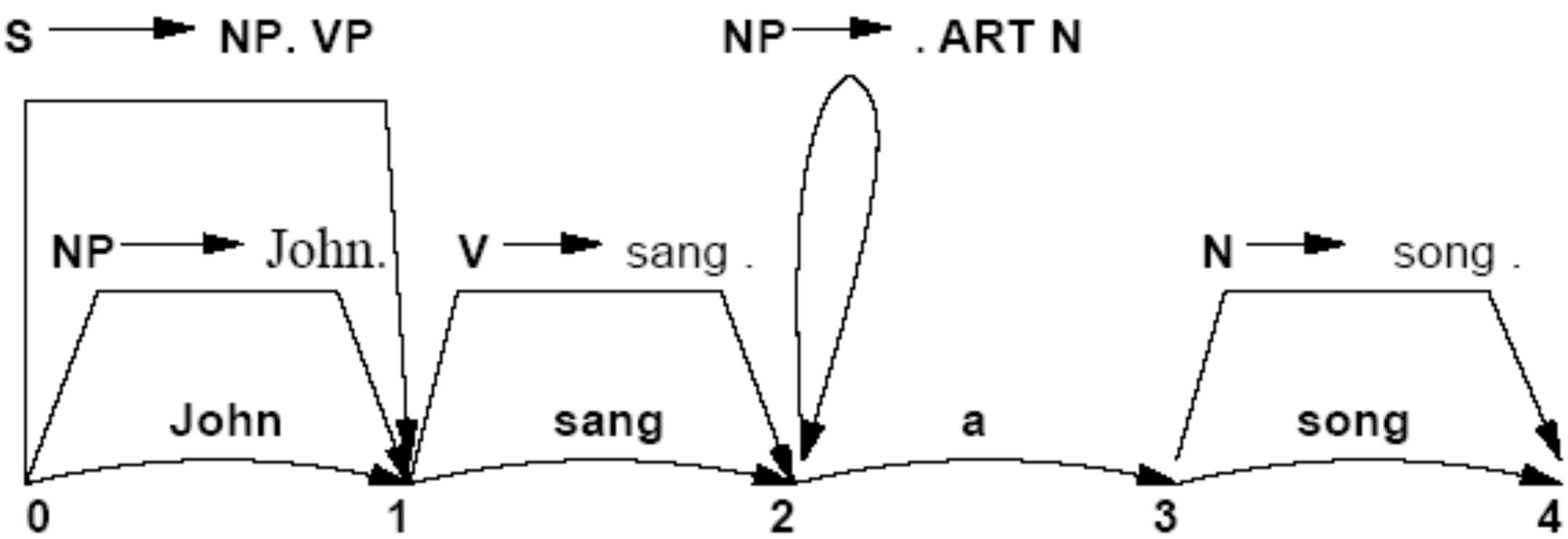
$$C \rightarrow X_1 \ldots X_j \bullet X_{j+1} \ldots X_k$$

Symbols on the left of the dot (•) have been already 'found' (confirmed hypotheses). Symbols on the right are still to be found.

Chart Parsing

$NP \rightarrow DET \bullet N$	**active edge**
$S \rightarrow \bullet\ NP\ VP$	**active and empty edge**
$VP \rightarrow V\ NP \bullet$	**inactive edge**

Chart Parsing



The chart has edges of the form

$$(i, j, A \rightarrow \alpha \bullet \beta)$$

Fundamental Rule of Chart Parsing

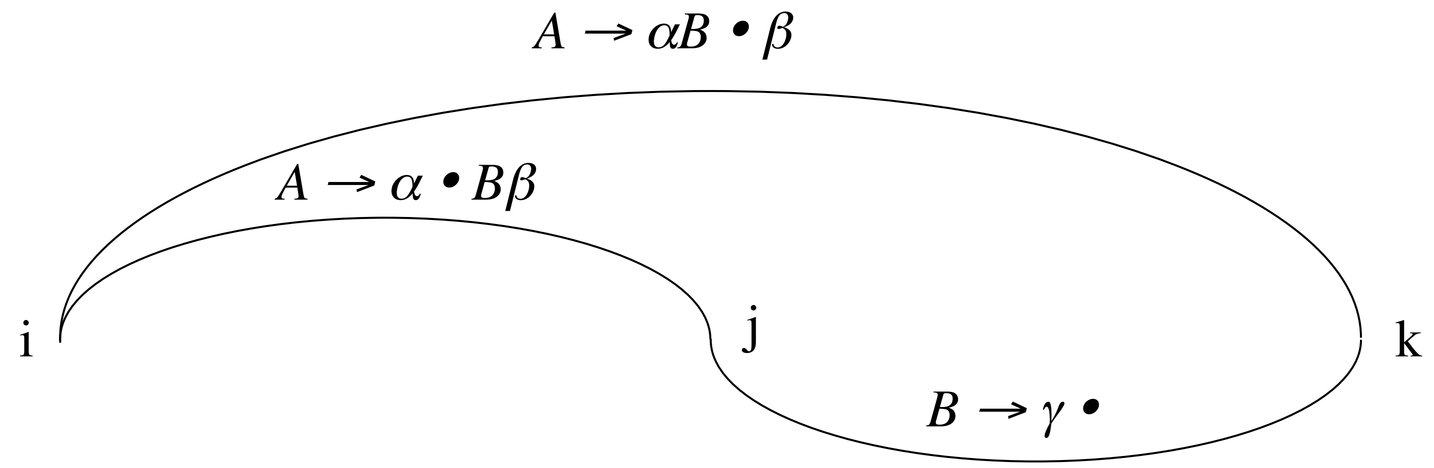
IF the chart contains the edges:

$$(i, j, A \rightarrow \alpha \bullet B\beta) \text{ and } (j, k, B \rightarrow \gamma \bullet)$$

THEN add the new edge:

$$(i, k, A \rightarrow \alpha B \bullet \beta)$$

(α, β, γ possibly empty strings of symbols)



Fundamental Rule of Chart Parsing

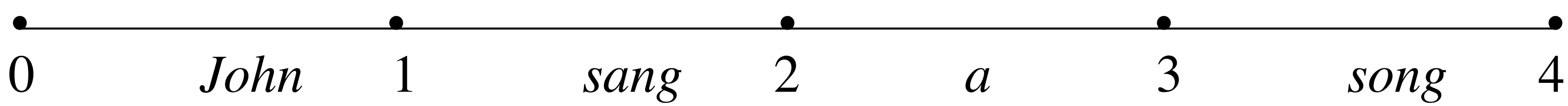
Fundamental rule only applies to chart containing active and inactive edges.

- How do we get started

Initialization:

Initially chart contains inactive edges corresponding to words in the input string:

e.g. for input *John sang a song*



Rule Invocation

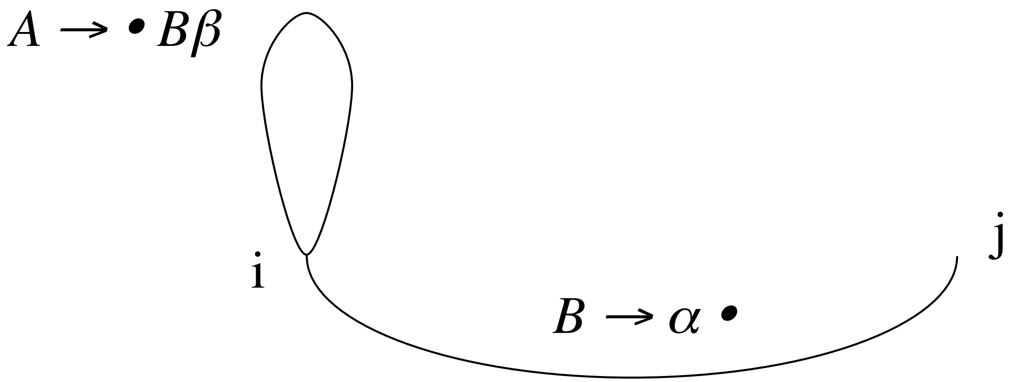
Bottom-Up :

IF you add an edge $(i, j, B \rightarrow \alpha \bullet)$

THEN for every rule of the form

$$A \rightarrow B\beta$$

add an edge $(i, i, A \rightarrow \bullet B\beta)$



Rule Invocation

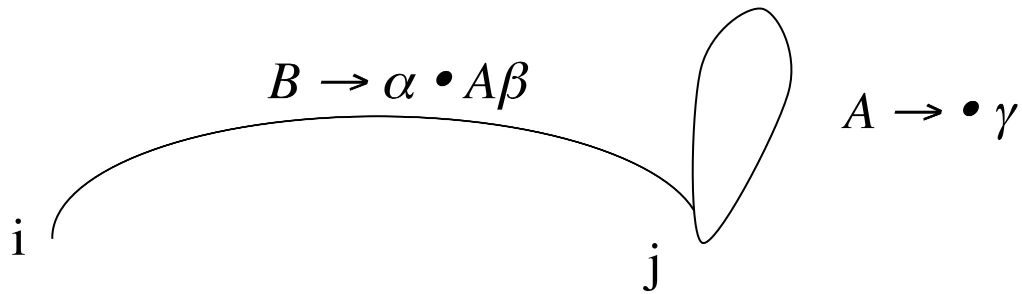
Top-Down:

IF you add an edge $(i, j, B \rightarrow \alpha \bullet A\beta)$

THEN for every rule of the form

$$A \rightarrow \gamma$$

add an edge $(j, j, A \rightarrow \bullet \gamma)$



Comparing CFGs and FSAs

FSAs:

- recognition is efficient – linear time; but
- the formalism is not very expressive.

CFGs:

- the basic parsing (recognition) strategies are not efficient – exponential time; but
- using dynamic programming techniques we can do better than this (Chart parsing; CKY algorithm; Earley algorithm); and
- CFGs are more expressive than FSAs

Comparing CFGs and FSAs

- Any language describable with a FSA is describable with a CFG.
- There are languages that can be described with a CFG that cannot be described with a FSA.

  Regular $\subset$ Context Free
- There is a general agreement that NLs are not Regular languages (i.e. cannot be adequately described with FSAs)
- Much of the syntax of the world's NLs seems to be Context Free (i.e. can be adequately described with CFGs).

Pros and Cons of CFGs

Advantages:

- Can describe infinite languages and assign appropriate syntactic structures
- Recognition (parsing) procedures can be implemented reasonably efficiently – $O(n^3)$:
  - Earley algorithm (Chart Parsing)
  - Cocke-Kasami-Younger (CKY) algorithm
  - Tomita's generalized LR parser

Pros and Cons of CFGs

- NLs ≅ CFGLs?
  - Long-standing argument
  - Arguably some NLs are non-CFLs (e.g. Swiss German – Shieber 1985)

Pros and Cons of CFGs

Disadvantages:

- Difficult to capture certain NL phenomena appropriately/adequately/elegantly:
  - agreement
  - subcategorization
  - generalizations over word/constituent order
  - relationships between different sentence types
- Some NL phenomena appear to require greater mathematical expressivity (i.e. there is evidence that some NLs are not CFLs)

Grammar equivalence

- Two grammars are **weakly equivalent** if they generate the same language (i.e. the same set of strings)
- Two grammars are **strongly equivalent** if they generate the same language *and* they assign the same phrase structure to each sentence

Mildly context-sensitive grammars (e.g. TAGs, Tree Adjoining Grammars)

1. Context Free Grammars (CFGs)
2. Efficiency and Expressivity
3. **Features and Unification**
4. Dependency Grammars
5. Resolving Ambiguity
6. Treebanks and Evaluation

3. Features and Unification

- Limitations of CFGs
- Unification-Based Grammars
  - Feature Structures
  - Unification
  - The PATR Formalism
  - Typed Feature Structures

Agreement phenomena

Verbs have to “agree” with subjects

NP	VP	
the boy	*sees the girl(s)*	singular
the boys	*see the girl(s)*	plural

Agreement phenomena

S → NPs VPs	VPs → Vs NPs
S → NPp VPp	VPs → Vs NPp
NPs → DETs Ns	VPp → Vp NPs
NPp → DETp Np	VPp → Vp NPp
DETs → *the*	Vs → *sees*
DETp → *the*	Vp → *see*
Ns → *boy*	Ns → *girl*
Np → *boys*	Np → *girls*

Subcategorization

Different verbs may require different complements

VP $\rightarrow$ V1	(*die*)
VP $\rightarrow$ V2 NP	(*love*)
VP $\rightarrow$ V3 NP NP	(*give*)
VP $\rightarrow$ V4 NP PP	(*put*)
VP $\rightarrow$ V5 NP S	(*tell*)
VP $\rightarrow$ V6 S	(*believe*)
and so on…	

Unbounded Dependency Constructions

E.g. Wh-questions:

Who did Bill see ε?

Who did Tom say that Bill saw ε?

Who did Anna believe Tom said that Bill saw ε?

- Correct interpretation of *who* depends on structure which is arbitrarily distant
- Difficult to capture UDCs with simple CFGs

Criteria for Formalism Design in NLP

Generative Power: can the formalism describe the language at all?

Notational Expressivity: can the formalism capture the appropriate generalizations?

Computational Effectiveness: does the formalism have a sensible, practical computational interpretation?

Note: simple CFGs score quite well on the first and third criteria; less well on the second.

Unification-Based Grammars

A family of related grammar formalisms.

UBGs can be viewed as extensions of CFGs which

- make use of constraints on feature values (to capture agreement, etc.)
- make use of syntactic features and allow underspecification of linguistic objects (categories or other representations)
- employ unification as a consistency checking / information merging operation

Examples of UBGs

- FUG (Kay)
- LFG (Bresnan & Kaplan)
- GPSG (Gazdar, Klein, Pullum & Sag)
- HPSG (Pollard & Sag)
- PATR (Shieber)
- CUG (Uszkoreit)
- UCG (Calder et el.)
- DUG (Hellwig)
- RUG (Carlson)
- TUG (Popovich)

Feature Structures

UBGs employ record-like objects to represent categories.

Third person singular NP:

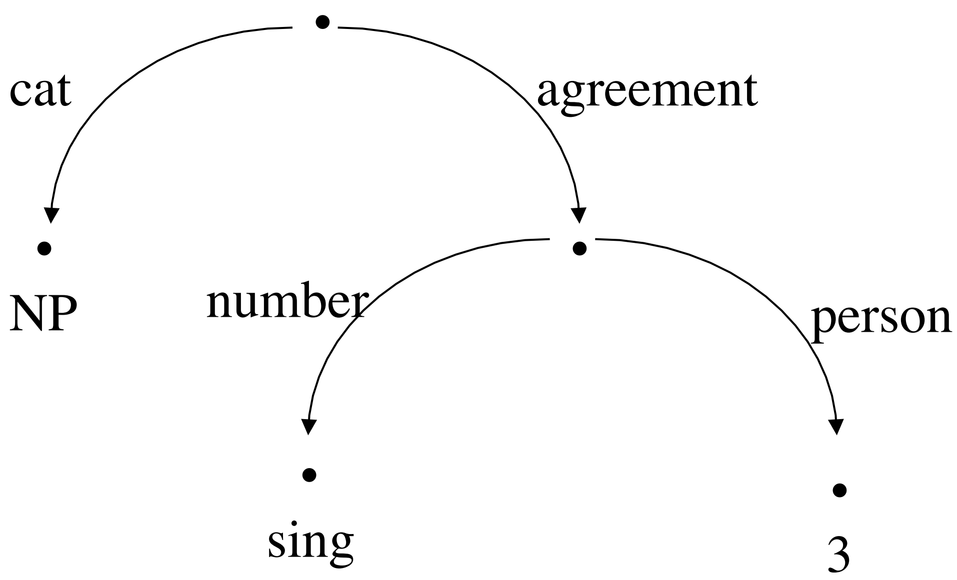
$$\begin{bmatrix} \textsc{cat} & \text{NP} \\ \textsc{agreement} & \begin{bmatrix} \textsc{number} & \text{sing} \\ \textsc{person} & 3 \end{bmatrix} \end{bmatrix}$$

- made up of features (`cat`, `agreement`, `number`, `person`) and values
- values may be *simple* (e.g., `NP`, `sing` and `3`) or *complex*:

$$\begin{bmatrix} \textsc{number} & \text{sing} \\ \textsc{person} & 3 \end{bmatrix}$$

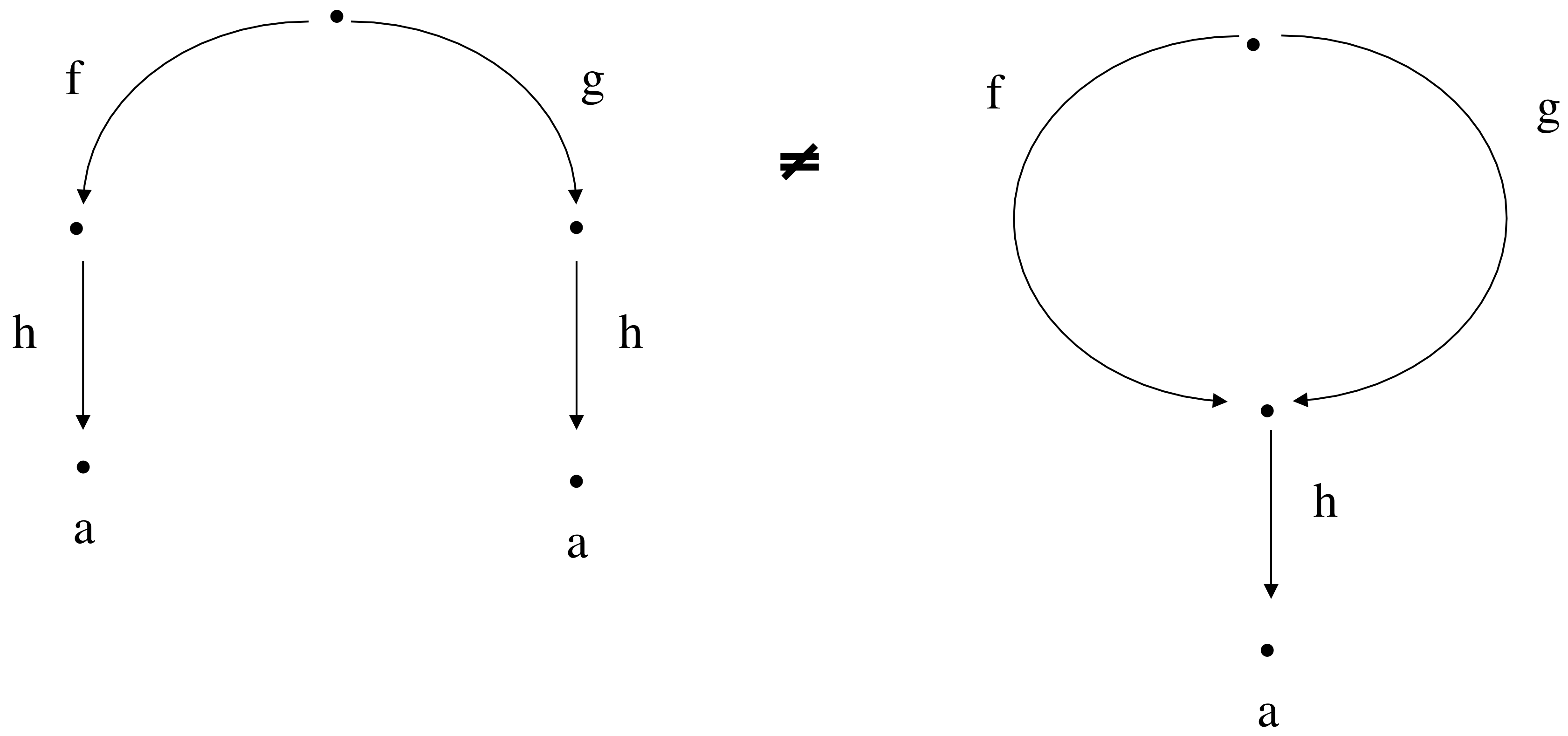
Feature Structures

Feature structures may be drawn as directed graphs:



Feature Structures

Feature structures may be *re-entrant*:



Feature Structures

Feature structures may be *re-entrant*:

$$\begin{bmatrix} \mathrm{F} & \begin{bmatrix} \mathrm{H} & \mathrm{a} \end{bmatrix} \\ \mathrm{G} & \begin{bmatrix} \mathrm{H} & \mathrm{a} \end{bmatrix} \end{bmatrix} \neq \begin{bmatrix} \mathrm{F} & \boxed{1} \begin{bmatrix} \mathrm{H} & \mathrm{a} \end{bmatrix} \\ \mathrm{G} & \boxed{1} \end{bmatrix}$$

Reentrant Feature Structures

A linguistic example:

$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \begin{bmatrix} \text{NUMBER} & \textsc{sg} \\ \text{PERS} & \textsc{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Feature Structures

Feature structures allow for underspecification of categories

Singular NP:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{sing} \end{bmatrix} \end{bmatrix}$$

Nominative NP:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{CASE} & \text{nom} \end{bmatrix}$$

Unification

F1 =

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{sing} \end{bmatrix} \end{bmatrix}$$

F2 =

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix} \\ \text{CASE} & \text{nominative} \end{bmatrix}$$

F1 $\cup$ F2 =

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{sing} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{CASE} & \text{nominative} \end{bmatrix}$$

Unification

Unification fails when feature structures are incompatible

$$\mathbf{F3} = \begin{bmatrix} \text{CAT} & \text{VP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \mathbf{sing} \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{F4} = \begin{bmatrix} \text{CAT} & \text{VP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \mathbf{plu} \end{bmatrix} \\ \text{VFORM} & \text{tensed} \end{bmatrix}$$

$$\mathbf{F3} \cup \mathbf{F4} = \mathbf{FAIL}$$

The PATR Formalism

Originally introduced by Shieber and his colleagues at SRI International

$S \longrightarrow NP\ VP$

$C_0 \longrightarrow C_1\ C_2$

$\langle C_0 \text{ cat}\rangle = \text{S}$

$\langle C_1 \text{ cat}\rangle = \text{NP}$

$\langle C_2 \text{ cat}\rangle = \text{VP}$

$\langle C_1 \text{ case}\rangle = \text{nominative}$

$\langle C_1 \text{ agreement}\rangle = \langle C_2 \text{ agreement}\rangle$

Typed Feature Structures

Limitations of simple feature structure formalisms:

- No way to constrain possible values of a feature (e.g., the feature `NUMBER` can take only `SING` and `PLU` values)
- No way to capture generalization across feature structures (e.g., different types of English verb phrases)

Solution: use of **types**.

Typed Feature Structures

- Each feature structure is labeled by a type
- Each type has **appropriateness conditions** expressing which features are appropriate for it
- Types are organized in a **type hierarchy**
- Unification should take into account the types of feature structures in addition to unifying attributes and values

1. Context Free Grammars (CFGs)
2. Efficiency and Expressivity
3. Features and Unification
4. **Dependency Grammars**
5. Resolving Ambiguity
6. Treebanks and Evaluation

4. Dependency Grammars

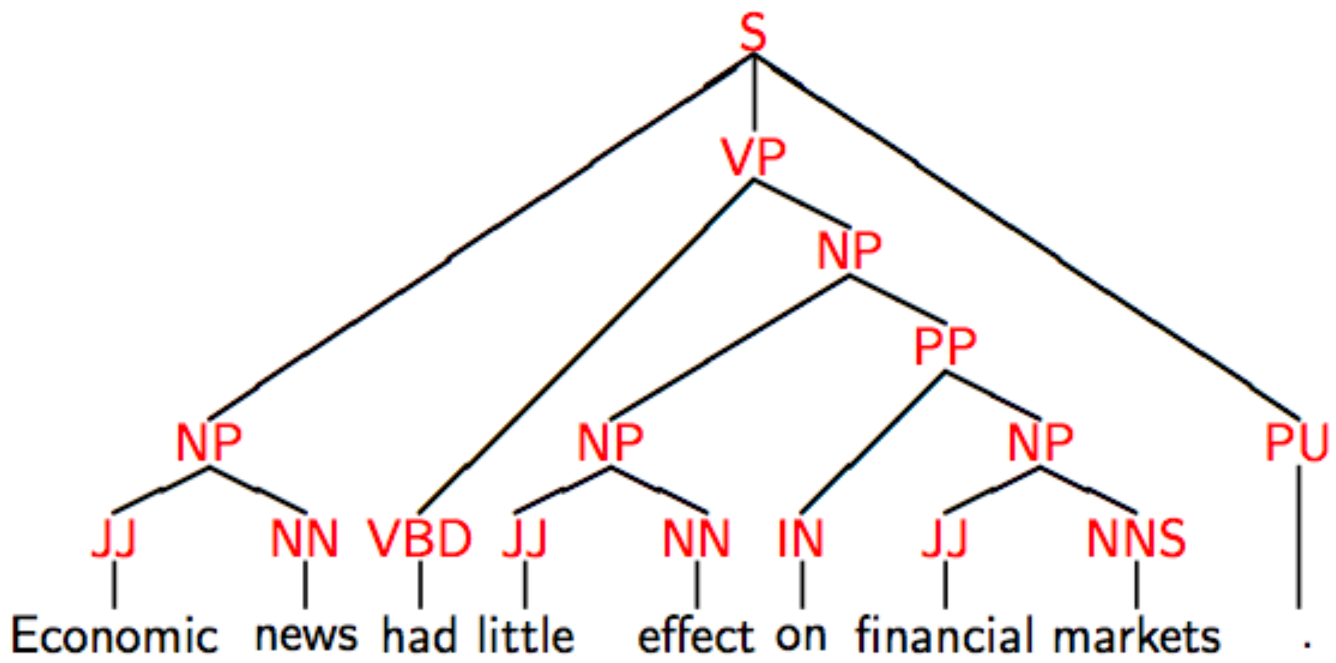
- Constituency vs. Dependency
- Dependency Grammars
- Dependency Parsing

some material taken from McDonald & Nivre ESSLLI 2007 course
“Introduction on Data-Driven Dependency Parsing”

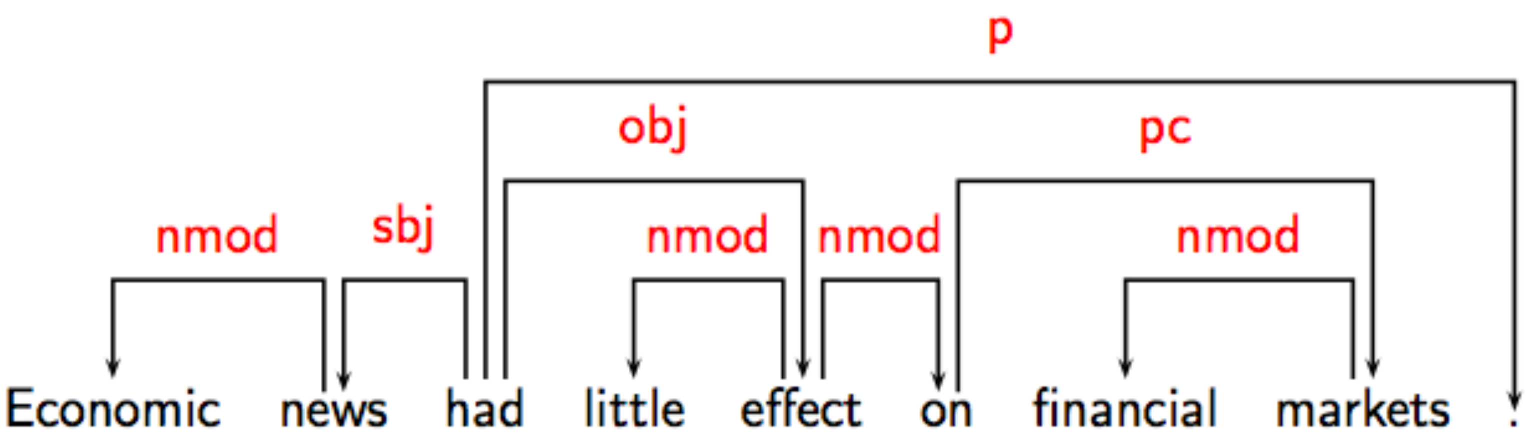
Dependency Grammars

- [Tesnière 1959]
- Syntactic structure of a sentence consists of lexical items, linked by binary asymmetric relations called dependencies.
- Dependency relations hold between a *head* (parent) and a *dependent* (daughter)

Phrase Structure



Dependency Structure



Constituency vs. Dependency

Phrase structure grammars

- Words appear only as leaves
- Internal nodes of trees consist of non-terminals

Dependency grammars

- No non-terminals
- Only words and binary relations between them

Constituency vs. Dependency

- Phrase structures explicitly represent
  - phrases (non-terminal nodes),
  - structural categories (non-terminal labels),
  - possibly some functional categories (grammatical functions).
- Dependency structures explicitly represent
  - head-dependent relations (directed arcs),
  - functional categories (arc labels),
  - possibly some structural categories (PoS).

Dependency Grammars

Family of grammatical formalisms differing in:

- Terminology (head/dependent, governor/ modifier, regent/subordinate, ...)
- Criteria adopted to establish dependency relations
- Criteria to identify heads and dependents

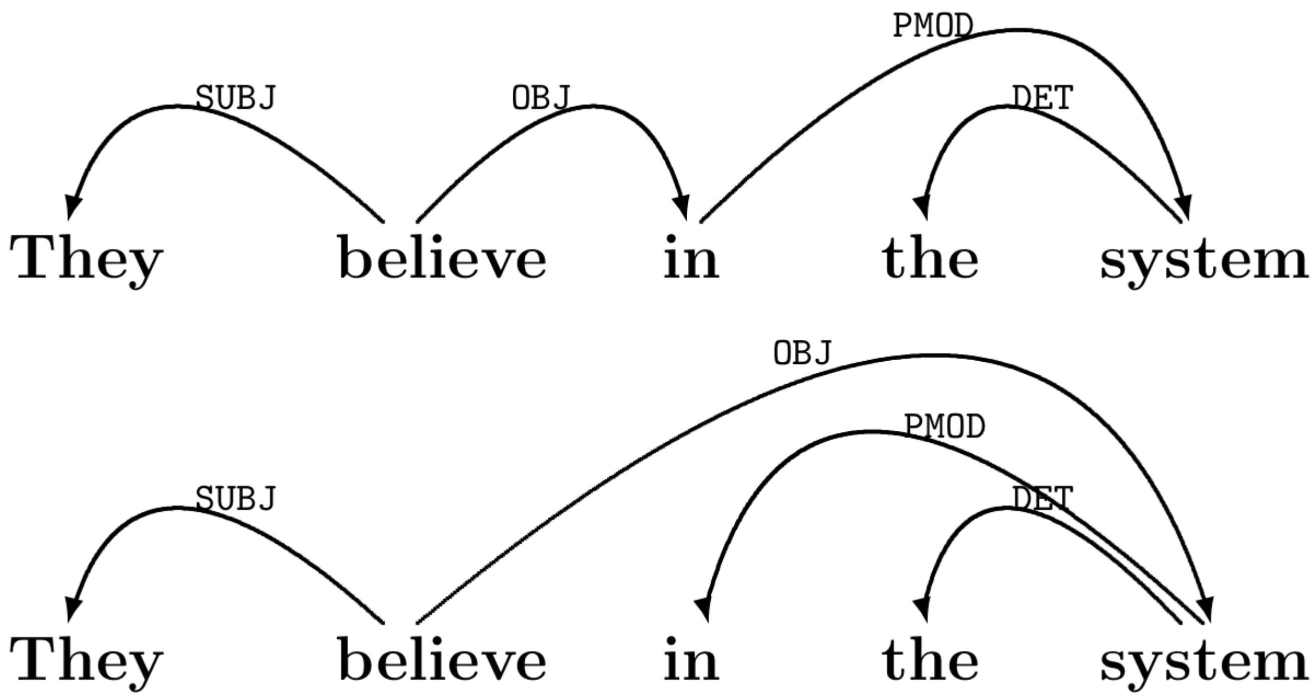
Dependency Relations

- Surface-oriented grammatical functions: *subject*, *object*, *adverbial*, …
- Semantically oriented roles: *agent*, *patient*, *goal*, …

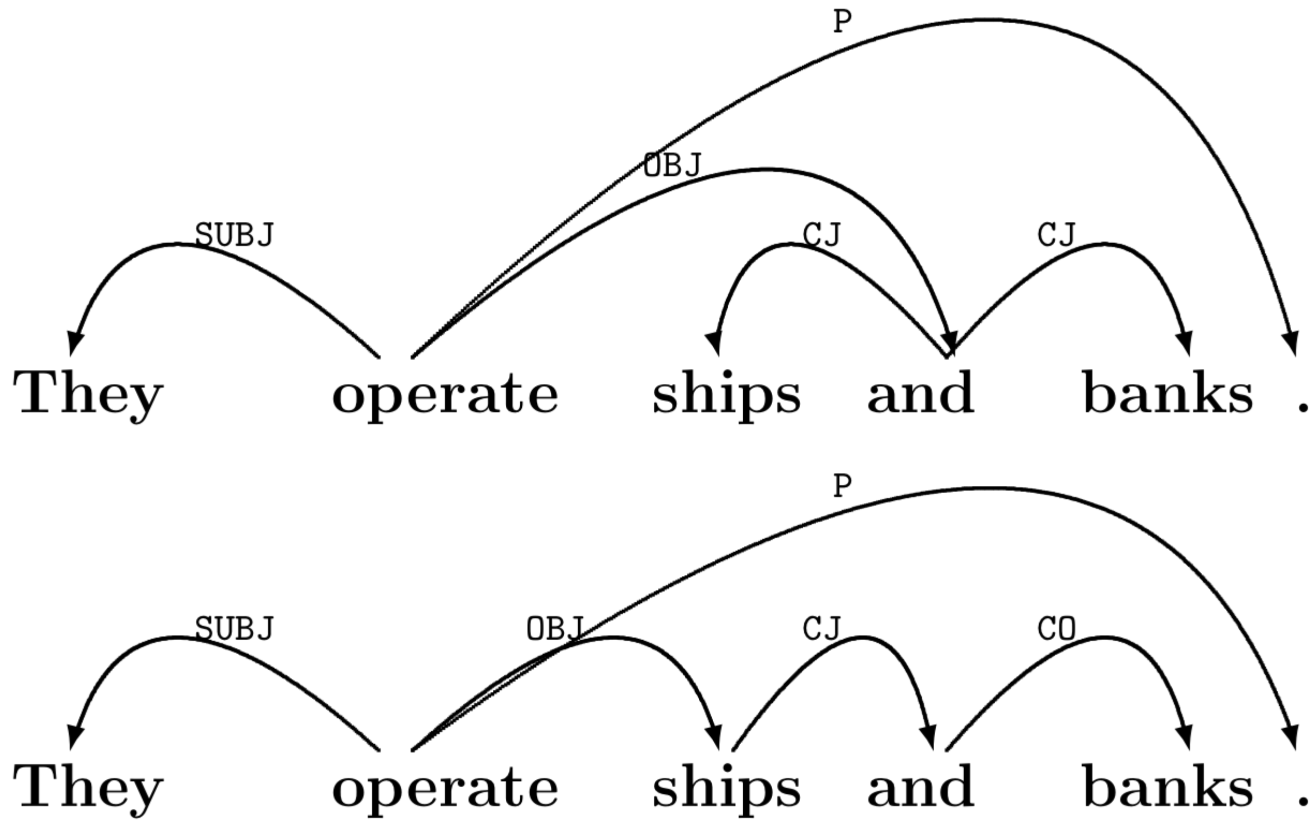
Problematic Constructions

- Grammatical function words: *syntactic* versus *semantic* heads
- Coordination: problematic in general

Function words



Coordination



Dependency Graphs

- A dependency structure can be defined as a directed graph *G*, consisting of
  - a set *V* of nodes (vertices),
  - a set *A* of arcs (directed edges),
  - a linear precedence order < on *V* (word order).

Dependency Graphs

- Labeled graphs:
  - Nodes in *V* are labeled with word forms (and annotation).
  - Arcs in *A* are labeled with dependency types:
    - $L=\{l_1, . . . , l_{|L|}\}$ is the set of permissible arc labels.
    - Every arc in *A* is a triple (i, j, k), representing a dependency from w_i to w_j with label l_k.

Dependency Parsing

- The problem:
  - Input: Sentence x=w_0, w_1, …, w_n with w_0=root
  - Output: Dependency graph $G = (V, A)$ for x where:
    - V={0, 1, . . . , n} is the vertex set,
    - A is the arc set, i.e., $(i, j, k) \in A$ represents a dependency from w_i to w_j with label $l_k \in L$