Dependency Grammar and Parsing

Dependency Parsing

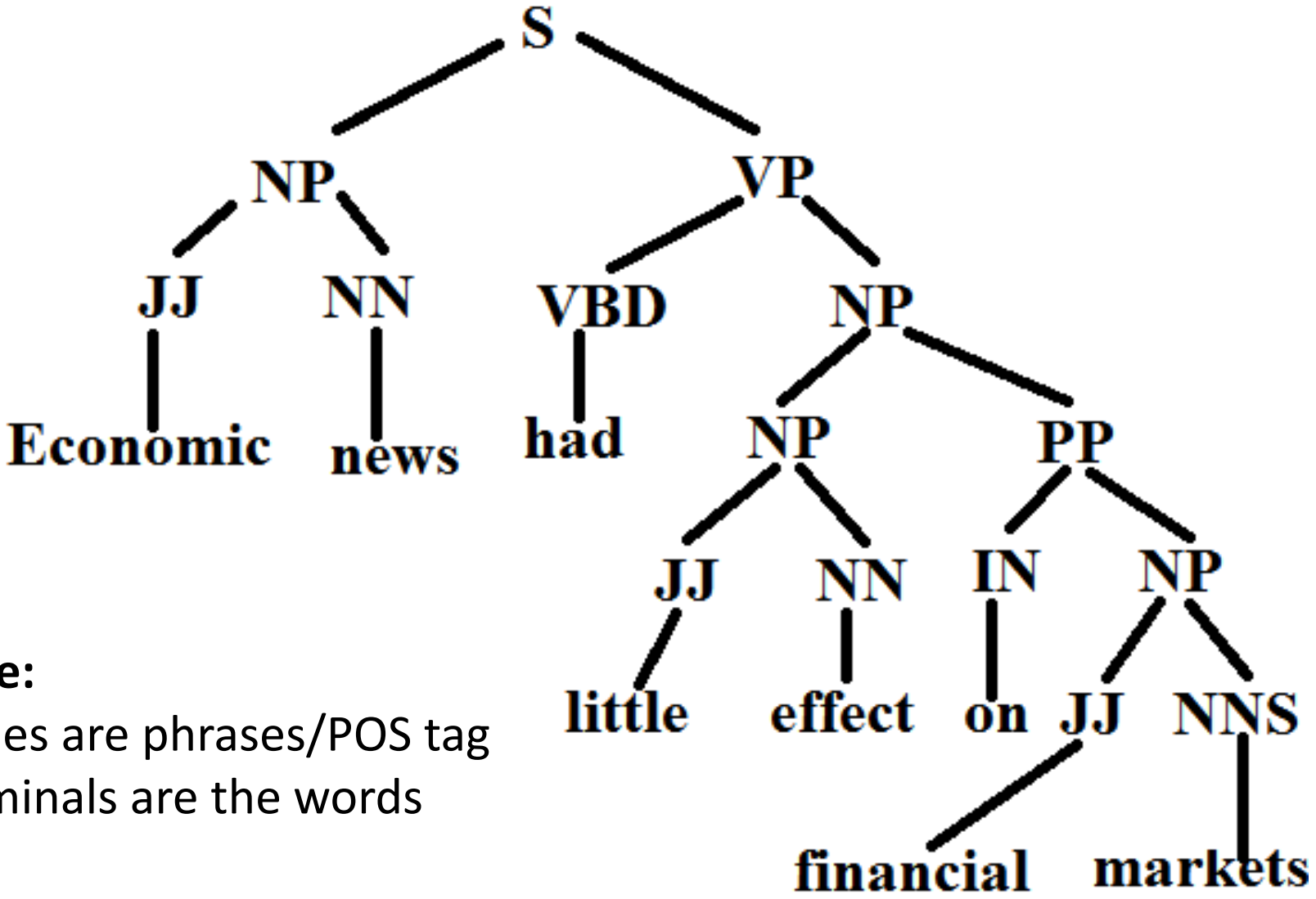
- In **Constituency/ Phrase structure**, we were finding what are the different word groups in terms of NP, VP, Adjective Phrase , Adverb phrase, etc.
- In **Dependency structure**, we will find out what is the relation between any two words.

Example: Ram ate apple

Main verb :- ate

What is the relation of Ram and apple with main verb ?

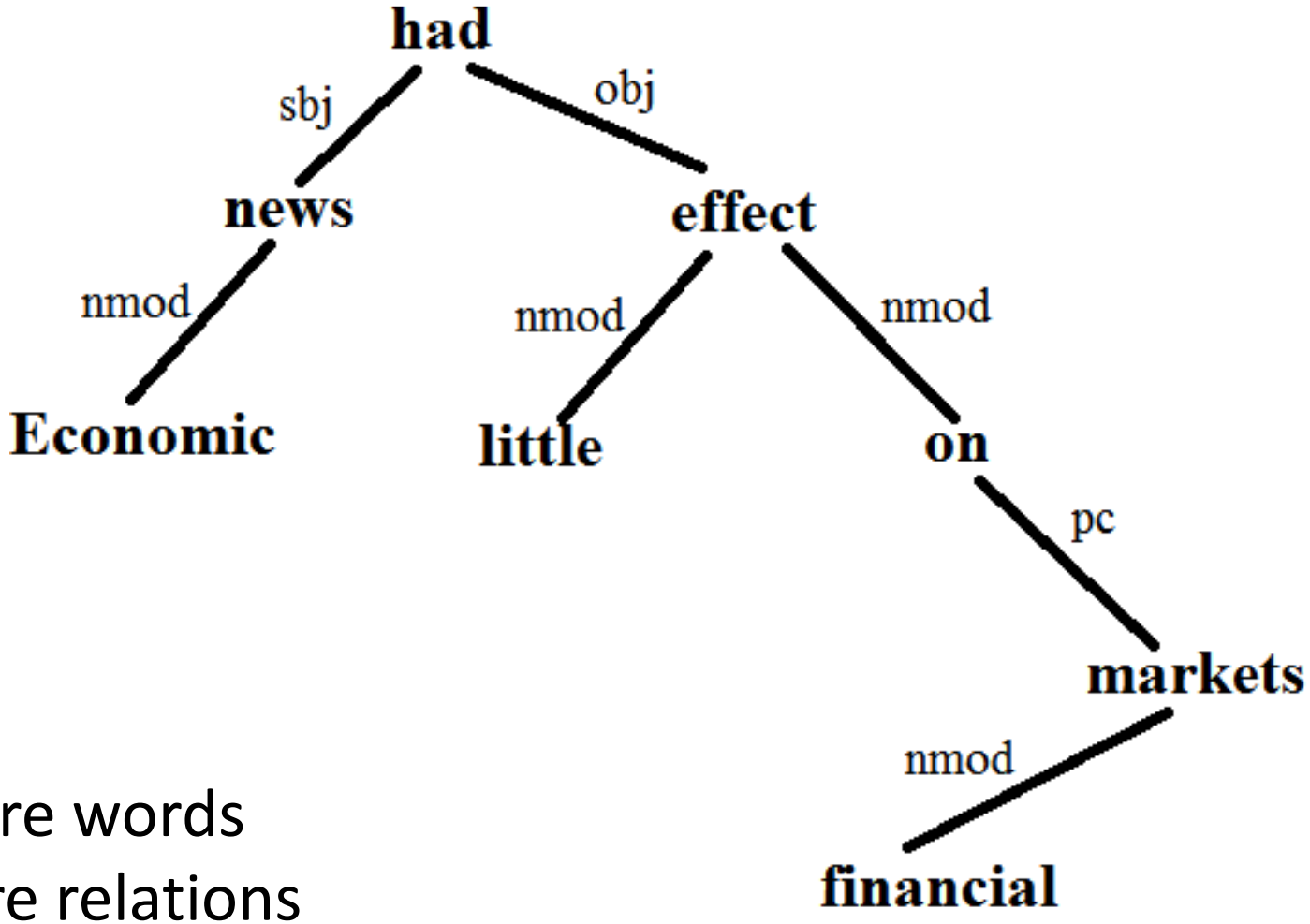
Phrase Structure/ Constituency Structure



It is a tree:

- Nodes are phrases/POS tag
- Terminals are the words

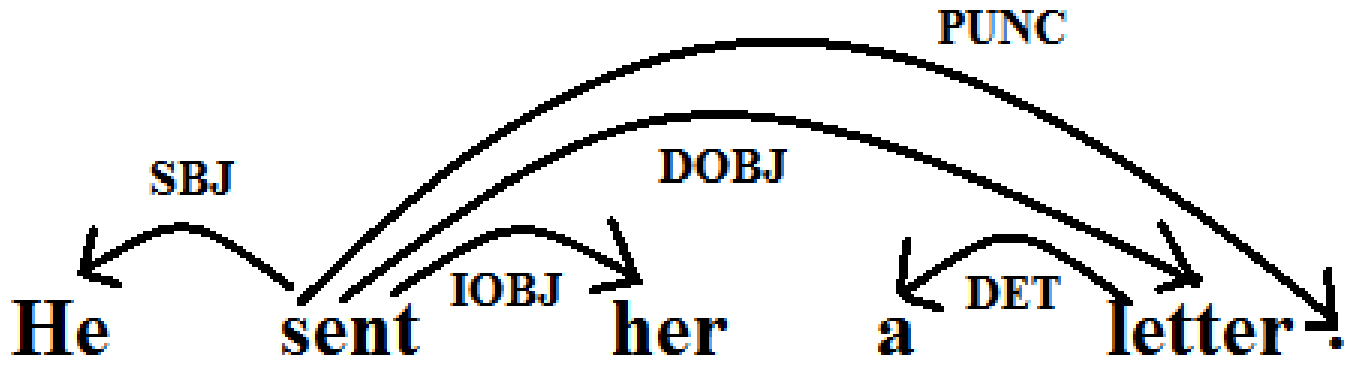
Dependency Structure



It is a tree:

- Nodes are words
- Edges are relations
- Two words are connected by a relation

Another Example



What are we doing in dependency structure

- Connects the words in sentence by putting arrows between the words.
  - Putting an arrow between **sent** and **He**. **He is a subject for sent.**
- Arrows show relations between the words and are typed by some grammatical relations.
- Arrows connect a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)
- Usually dependencies form a tree
- Arrow points from head to dependent

Dependency Parsing

Points to discuss:

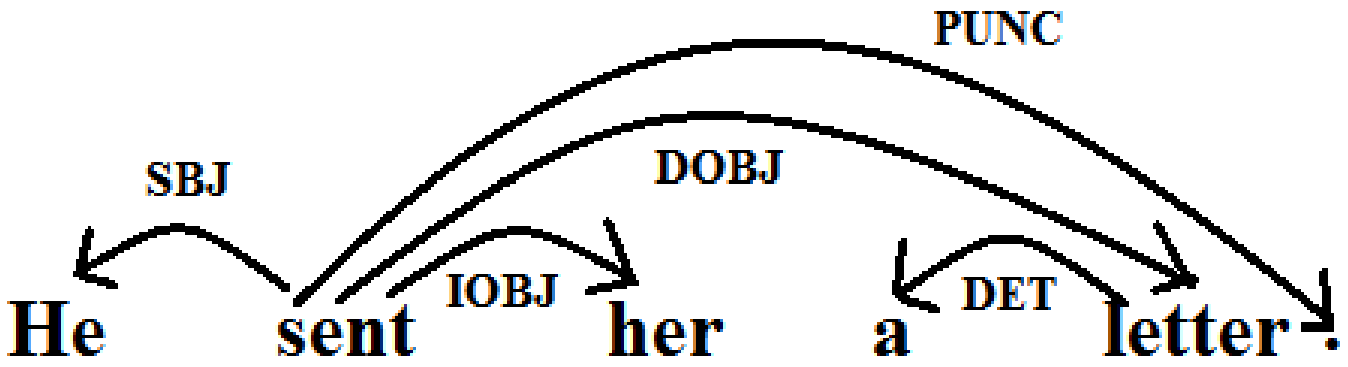
1. What are the formal ways of defining dependency structure?
2. What are the linguistic constrains that we need to impose on the structure?
3. What are the some of the popular models for getting the dependency structure starting from the sentence?

What is the Criteria for Heads and Dependents

Criteria for a syntactic relation between a head H and a dependent D in a construction C.

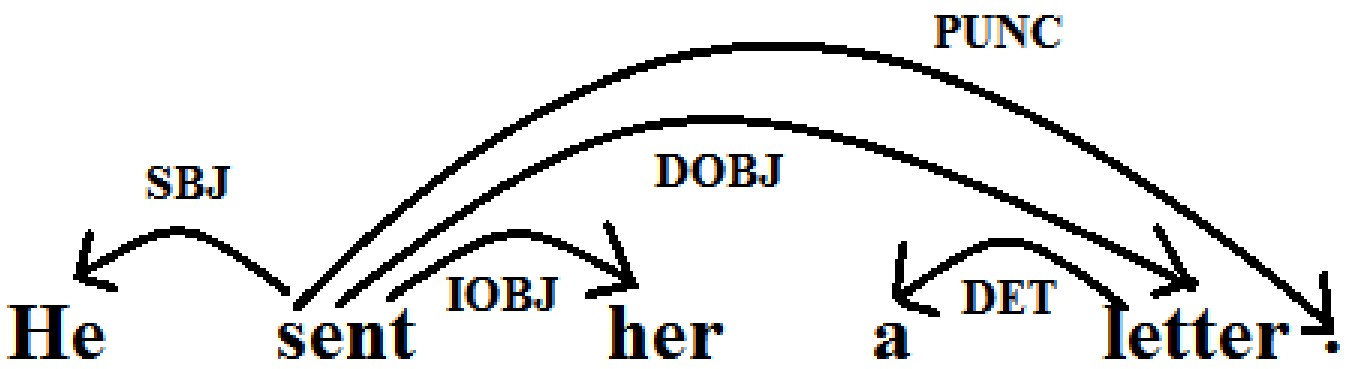
1. H determines the syntactic category of whole C, H can replace C
2. D determines/Specifies H
3. H is obligatory, D may be optional
4. H selects D and determines whether D is obligatory
5. The form of D depends on H
6. The linear position of D is specified with ref. to H

Criteria 1: H determines the syntactic category of whole C, H can replace C



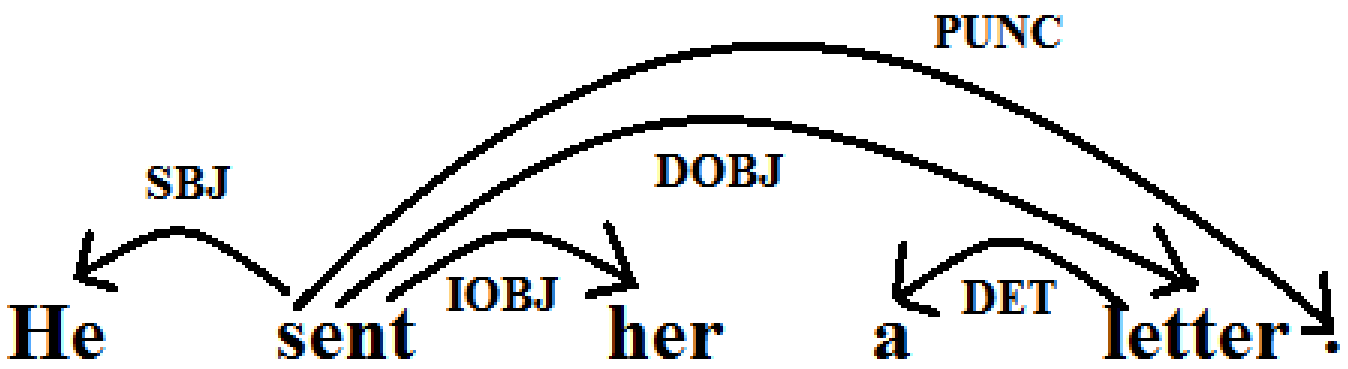
Example: if we see the construction **a letter**, the whole syntactic construct is **governed by** word **letter**, that is the **word letter becomes a head**, the whole construction **a letter** can be replaced by **letter**

Criteria 2: D determines H



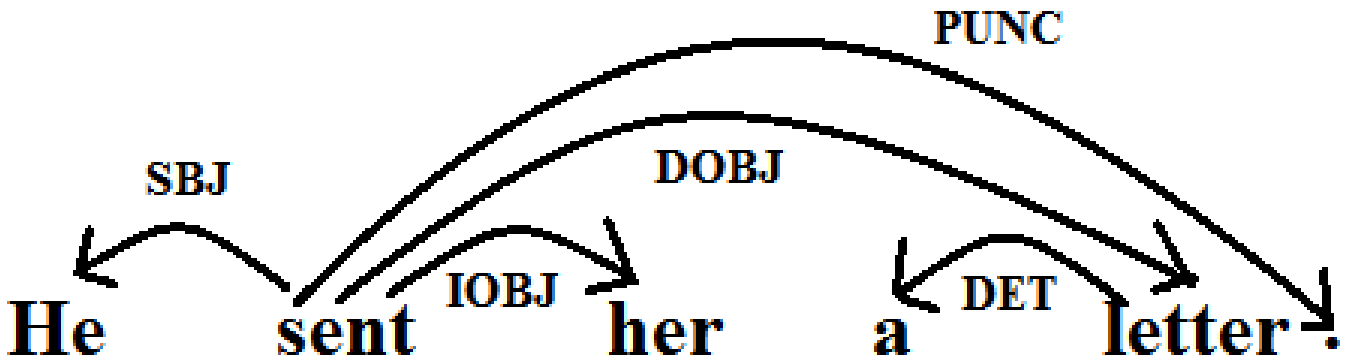
- Now, **D specifies H**, i.e. D is giving further specific information about H.
- So, if we say **only letter** than it may not be as specific as compared to **a letter**. So, all these determiners give some additional information.

Criteria 2: D Specifies H



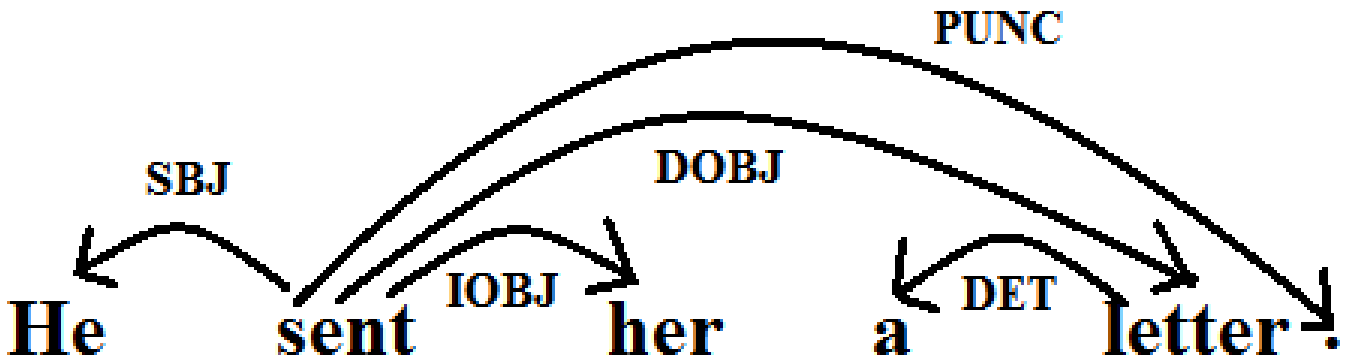
- Similarly, for **sent her a letter** , where sent is the head and letter is a modifier.
- If we simply say, **he sent**, then it is not very specific, to whom did he sent, there we have to specify particular modifier i.e., **he sent her .**
- So what did he sent, then we have to put the word "**a letter**".
- So what are you observing, the dependent is further specifying the head.

Criteria 3: H is obligatory, D may be optional



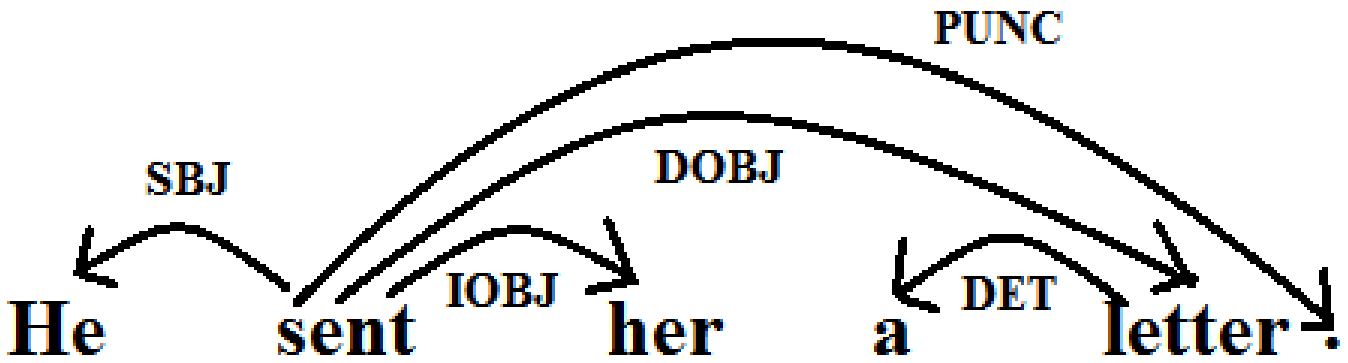
- Head is always obligatory, i.e. in some cases a letter is necessary
- But dependent may be optional in some case i.e, he sent her letter

Criteria 4: H selects D and determines whether D is obligatory



- In some cases, the grammatical form that D takes will also depend upon the head H. This is also called agreement or government in case of linguistic.

Criteria 5: The form of D depends on H



- The grammatical form used for head in construction will be the same to the form of subject or object in our sentence.

Criteria 6: The linear position of D is specified with reference to H

English Language follows this construct:

SVO i.e., subject verb object

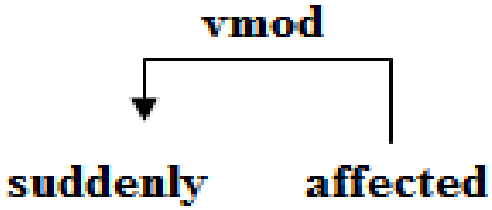
- The linear position of dependent is specified with respect to head
- If verb is main head then we know subject will come to the left and object will come to the right.
- Example: I am eating an apple
  - eat –main verb i.e, head
  - who is eating –will come to left
  - What do I eat – will come to the right

Some Cases where easily dependencies can be found

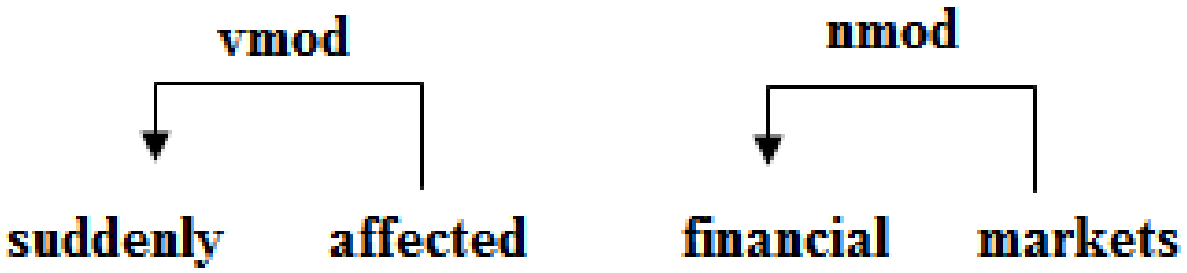
Construction	Head	Dependent
Exocentric	Verb	Subject (sbj)
	Verb	Object (Obj)
Endocentric	Verb	Adverbial (vmod)
	Noun	Attribute (nmod)

- **Endocentric:** It is one where one of the entity can actually fulfill the whole grammatical function of the complete construction.

Example: verb and adverbial relation (vmod)



continued....



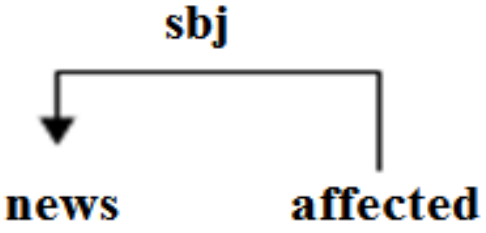
- For the verb-modifier - **suddenly affected**
- The word **affected alone** can fulfill the whole grammatical function for the whole construction “suddenly affected” where suddenly is simply modifying that.
- Similarly, for noun-modifier **- financial markets**
- The word **markets** can fulfill the whole grammatical function for the whole construction “financial markets” .

Some Cases where easily dependencies can be found

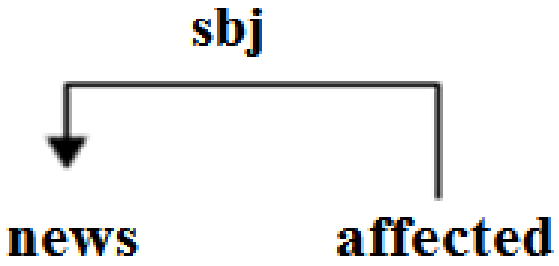
Construction	Head	Dependent
Exocentric	Verb	Subject (sbj)
	Verb	Object (Obj)
Endocentric	Verb	Adverbial (vmod)
	Noun	Attribute (nmod)

Exocentric: It is one where one of the entity cannot actually fulfill the whole grammatical function of the complete construction.

Example: verb and subject relation (sbj)



continued.....



- The word **affected** cannot fulfill the whole grammatical function for the whole construction “news and affected”. We need to have the word “**news”** . So this is exocentric.
- Thus, in endocentric one word can fulfill the whole function but exocentric cannot.

What will become head and what will become dependent ?

Construction	Head	Dependent
Exocentric	Verb	Subject (sbj)
	Verb	Object (Obj)
Endocentric	Verb	Adverbial (vmod)
	Noun	Attribute (nmod)

- **In case of endocentric**, a particular word that can fulfill the grammatical function will become head and the other dependent.
- **In case of exocentric**, if we have verb and object then verb will become head and object will become dependent.

Comparison: Phrase structure & Dependency structure representation

- Phrase Structure explicitly represent
  - phrases (non-terminal nodes)
  - structured categories (non-terminal nodes)
- Dependency Structure explicitly represent
  - Head-dependent relations (directed arcs)
  - Functional categories (arc labels)

Dependency Graphs

Dependency Graphs

- A dependency structure can be defined as a directed graph G, consisting of :
  - A set V of nodes
  - A set A of arcs (edges)
- Dependency graphs are labeled graphs:
  - Nodes in V are labeled with word forms (and annotations)
  - Arcs in A are labeled with dependency types
- Notational Conventions:

  Arc(w_i, d, w_j) links head w_i to dependent w_j with label d.

  $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
- Other conditions: $i \rightarrow j \equiv (i, j) \in A$

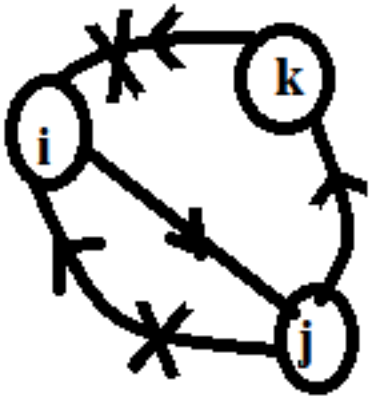
  $i \rightarrow^* j \equiv i = j \vee \exists k: i \rightarrow k , k \rightarrow^* j$

Formal Conditions on Dependency Graphs

1. G is connected
2. G is acyclic
3. G obeys the single head constrain
4. G is projective

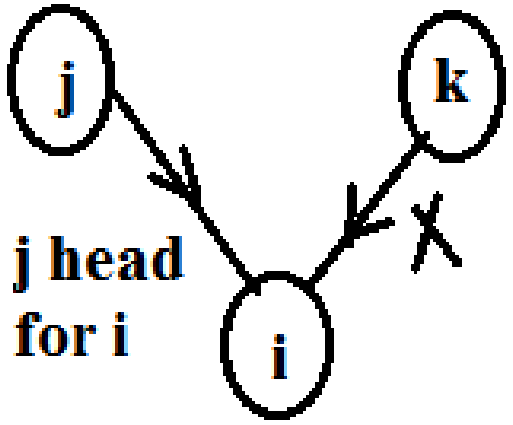
Formal Conditions on Dependency Graphs

1. G is connected:
   - No node in the graph is an isolated node
   - For every node i there is a node j such that i $\longrightarrow$ j or j $\longrightarrow$ i
2. G is acyclic:
   - If i $\rightarrow$ j then not j $\rightarrow$ *i
   - From label i there is a path to j, then there is no way to go back to i.



Formal Conditions on Dependency Graphs

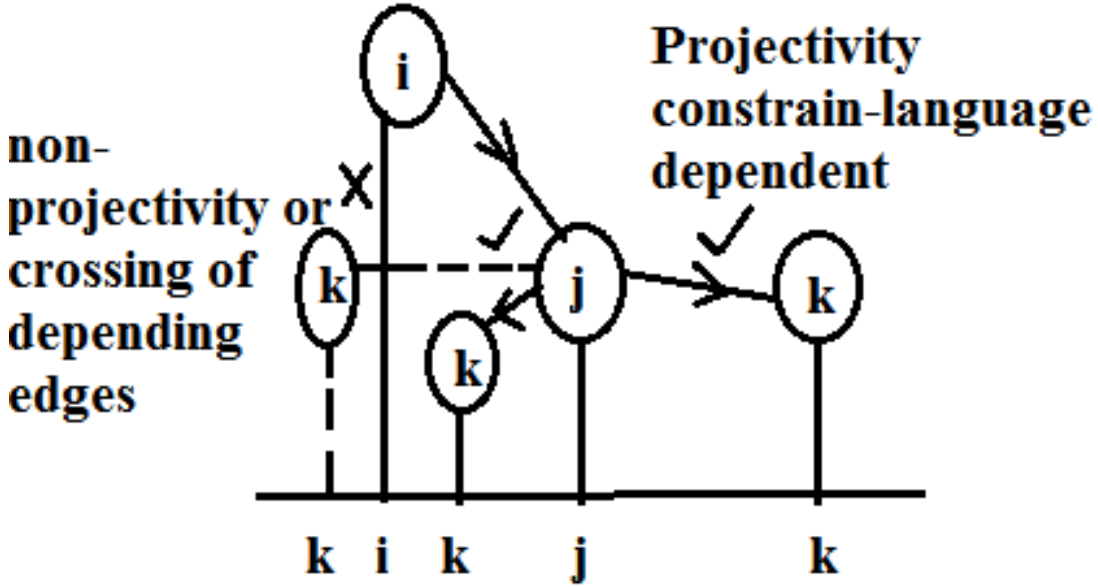
3. G obeys the single head constrain:
   - If i → j then not k → j, for any k ≠ i
   - If node i is in graph G, then there can be atmost one head for it.



If j is head for i then k cannot be head of i.

Formal Conditions on Dependency Graphs

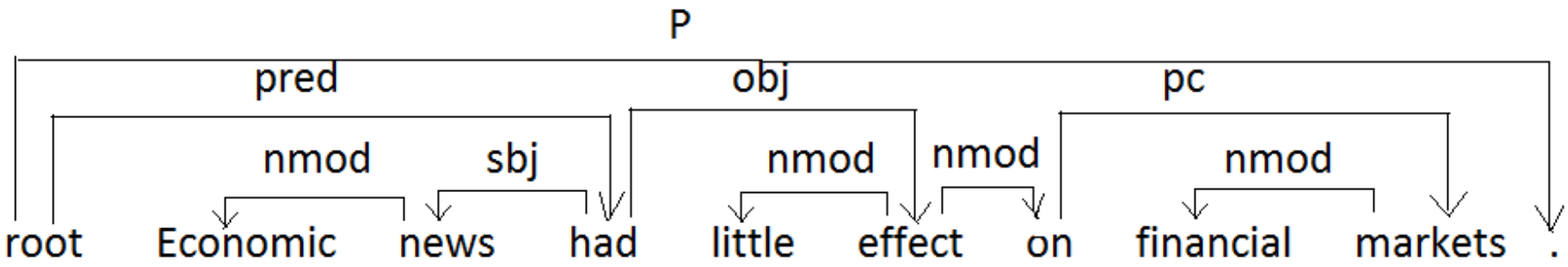
3. G is projective
   - If i → j then j → *k, for any k such that both j and k lie on the same side of i
   - It says, if from i we can derive j and from j can derive k then for any k such that j to k lie on the same side of i.



Formal Conditions: Basic Intuition

1. Connectedness: Syntactic structure is complete.
2. Acyclicity: Syntactic structure is hierarchical.
3. Single-head: Every word has at most one syntactic head.
4. Projectivity: No crossing of dependencies

Connectedness can be enforced by adding a special root node.

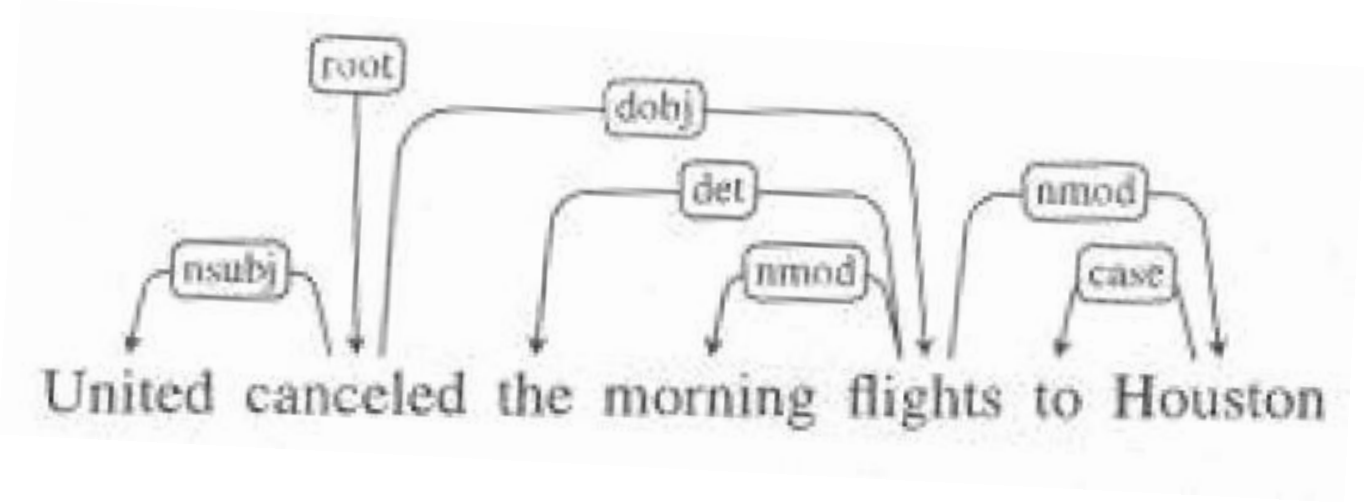


Dependency Relations

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marn-

Consider the following sentence:



- The clausal relations : **NSUBJ** and **DOBJ** identify the **subject** and **direct object** of the predicate **cancel**,
- The NMOD, DET and CASE relations denote modifiers of the nouns flights and Houston

Dependency Parsing

Dependency Parsing

Dependency Parsing

- **Input** : Sentence x = w_1, …..w_n
- **Output**: Dependency Graph G that follows all constrains

Parsing Methods

- Dynamic Programming
- Constrain Propagation Based
- Deterministic Parsing
  - Transition based Parsing

Deterministic Parsing

Basic idea:

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

Configurations:

A parser configuration is a triple c = (S,B,A), where

- S : a stack $[\ldots, w_i]_S$ of partially processed words,
- B : a buffer $[w_j, \ldots]_B$ of remaining input words,
- A : a set of labeled arcs (w_i, d, w_j)

Stack	Buffer	Arcs
$[\text{sent, her, a}]_S$	$[\text{letter, .}]_B$	He $\overset{\text{SBJ}}{\leftarrow}$ sent

Transition System

A transition system for dependency parsing is a quadruple: $S = (C, T, c_s, C_t)$,

where

- C is a set of configurations,
- T is a set of transitions, such that $t : C \rightarrow C$,
- c_s is an initialization function
- $C_t \subseteq C$ is a set of terminal configurations.

A transition sequence for a sentence x is a set of configurations

$C_{0,m} = (c_0, c_1, \ldots, c_m)$ such that

$c_0 = c_s(x)$, $c_m \in C_t$, $c_i = t(c_{i-1})$ for some $t \in T$

Initialization: $([]_S, [w_1, \ldots, w_n]_B, \{\})$

Termination: $(S, []_B, A)$

Transitions for Arc-Eager Parsing

$$\text{Left-Arc}(d)\ \frac{([\ldots, w_i]_S,\ [w_j, \ldots]_B,\ A)}{([\ldots]_S,\ [w_j, \ldots]_B,\ A \cup \{(w_j, d, w_i)\})}\ \neg\text{HEAD}(w_i)$$

$$\text{Right-Arc}(d)\ \frac{([\ldots, w_i]_S,\ [w_j, \ldots]_B,\ A)}{([\ldots, w_i, w_j]_S,\ [\ldots]_B,\ A \cup \{(w_i, d, w_j)\})}$$

$$\text{Reduce}\ \frac{([\ldots, w_i]_S,\ B,\ A)}{([\ldots]_S,\ B,\ A)}\ \text{HEAD}(w_i)$$

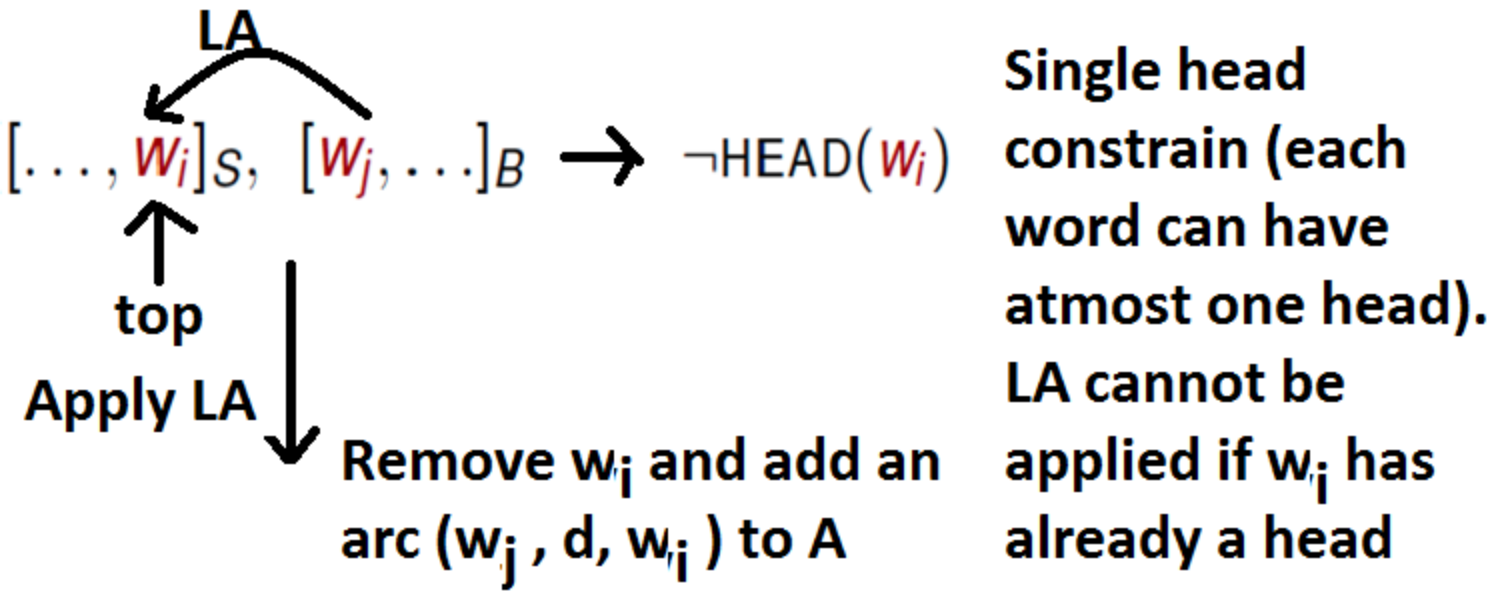
$$\text{Shift}\ \frac{([\ldots]_S,\ [w_i, \ldots]_B,\ A)}{([\ldots, w_i]_S,\ [\ldots]_B,\ A)}$$

For any configuration, take one of these four transitions- left-arc, right-arc for a dependency d, reduce and shift.

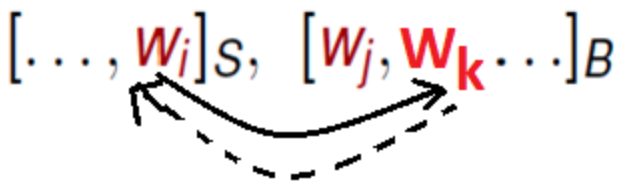
$$\text{Left-Arc}(d)\ \frac{([\ldots, w_i]_S,\ [w_j, \ldots]_B,\ A)}{([\ldots]_S,\ [w_j, \ldots]_B,\ A \cup \{(w_j, d, w_i)\})}\ \neg\text{HEAD}(w_i)$$

Left Arc transition (Left-Arc(d)):

- LA(left-arc) will be from buffer to stack.
- Can apply it when stack contains some word w_i (top of stack)
- Buffer contains some word w_j,
- Refer dependency graph
- left-arc means making a transition from w_j to w_i, because the words are occurring in the same order that they occur in the sentence.
- w_j is the head and w_i is the dependent.
- There is a condition, w_i should not already have a head.



- w_i can only be removed if all its relations have already been captured.
- Can w_i have any relations with words before w_i . The words before w_i are partially processed words, so there relations are already captured.
- Now, w_i might have relation with words after w_j e.g., w_k
- What kind of relation w_i might have with w_k?
  - It might be incoming arc or outgoing arc.



- It cannot have incoming arc from w_k to w_i as more than one head.(single head constrain), so incoming arc will not exist.
- Now, outgoing arc $w_i \rightarrow w_k$,
- Remember formal conditions, it is violating the projectivity constrain. If there is a relation from j to i then from i next dependent would be side of i and not side of j. That means w_i cannot have any further relations, so now w_i can be removed from stack and add arc $\{w_j, d, w_i\}$ to A.

$$[\ldots\ldots]_S,\ [w_j,\ ..]_B \quad A \cup \{(w_j, d, w_i)\}$$

Right Arc transition:

$$\text{Right-Arc}(d)\ \frac{([\ldots, w_i]_S,\ [w_j, \ldots]_B,\ A)}{([\ldots, w_i, w_j]_S,\ [\ldots]_B,\ A \cup \{(w_i, d, w_j)\})}$$

- Right-arc, relation from w_i to w_j, w_i is the head and w_j is the dependent.
- When you apply right arc transition, w_j goes to the stack S.
- Do not remove w_i from stack and w_j goes to the S stack and is new top and we get a new arc $\{w_i, d, w_j\}$ added to A.

Reduce transitions

$$\text{Reduce}\ \frac{([\ldots, w_i]_S,\ B,\ A)}{([\ldots]_S,\ B,\ A)}\ \text{HEAD}(w_i)$$

- In reduce transition, we remove the top word, w_i, from the S stack and the condition is w_i already has a head.
- If w_i already has a head, we can remove that.

Shift transitions

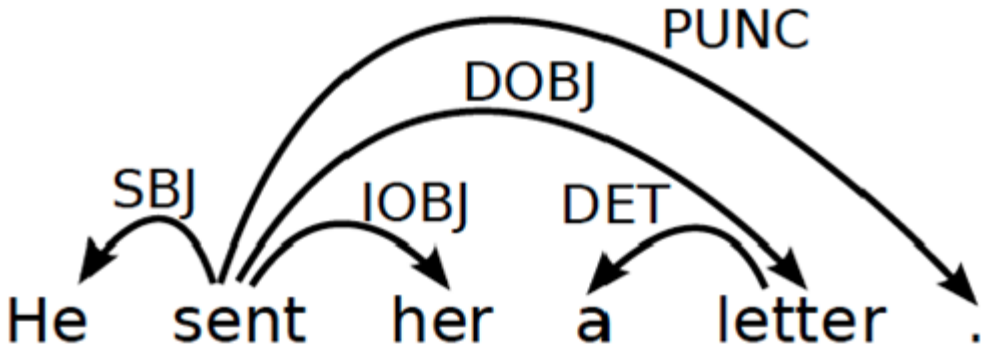
$$\text{Shift}\ \frac{([\ldots]_S,\ [w_i,\ldots]_B,\ A)}{([\ldots,w_i]_S,\ [\ldots]_B,\ A)}$$

- The fourth transition, shift, takes the top word on the buffer and shift it to the top of S stack.

Example: Given a sentence and its dependency parse.

Sentence : He sent her a letter.

Dependency parse:



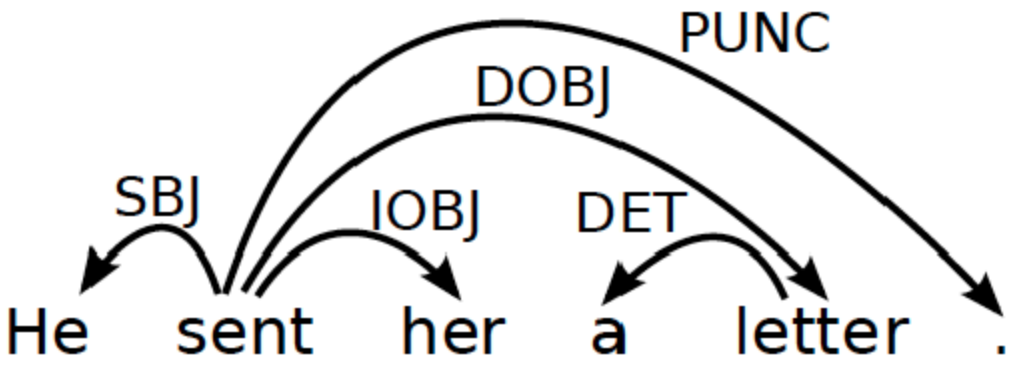
Start with finding initial configuration

Stack	Buffer	Arcs
$[\,]_S$	$[\text{He, sent, her, a, letter, .}]_B$	{ }

Example:

Transitions: SH

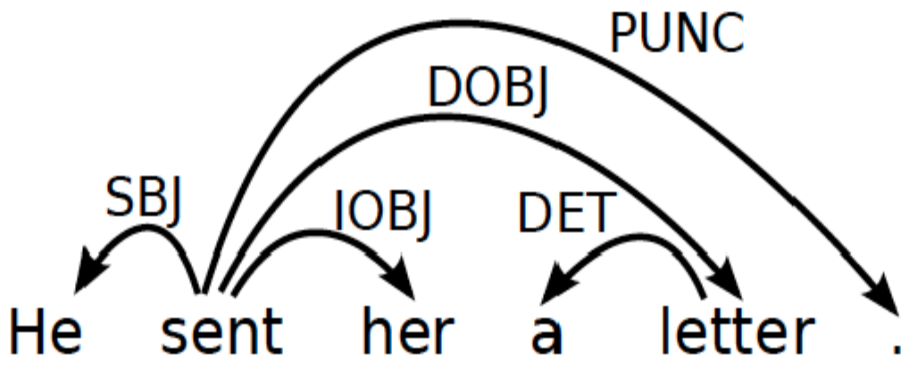
Stack	**Buffer**	**Arcs**
$[\text{He}]_S$	$[\text{sent, her, a, letter, .}]_B$	



Example:
Parse Example

Transitions: SH-LA

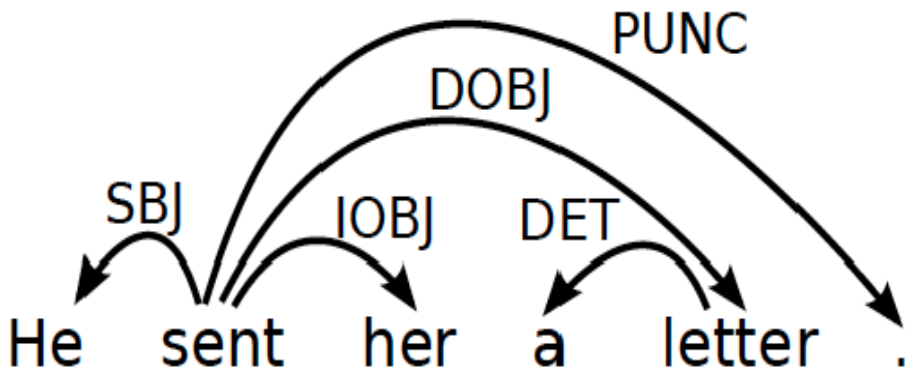
Stack	Buffer	Arcs
$[\ ]_S$	$[\text{sent, her, a, letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent



Example:
Parse Example

Transitions: SH-LA-SH

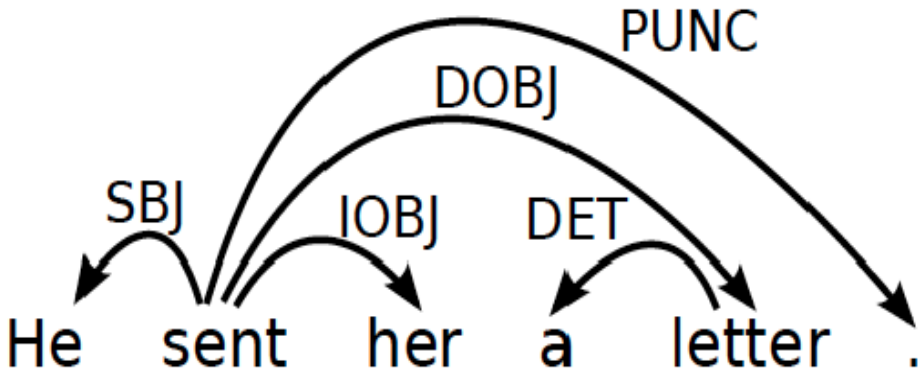
Stack	Buffer	Arcs
$[\text{sent}]_S$	$[\text{her, a, letter, .}]_B$	He $\overset{\text{SBJ}}{\longleftarrow}$ sent



Example:
Parse Example

Transitions: SH-LA-SH-RA

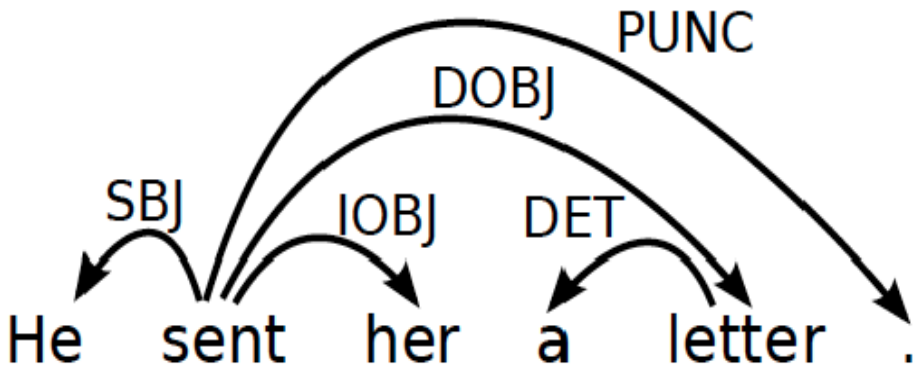
Stack	Buffer	Arcs
$[\text{sent, her}]_S$	$[\text{a, letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her



Example:
Parse Example

Transitions: SH-LA-SH-RA-SH

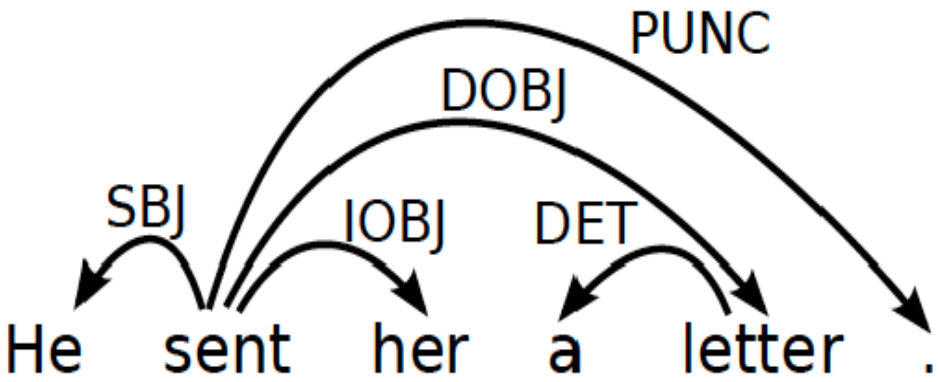
Stack	Buffer	Arcs
$[\text{sent, her, a}]_S$	$[\text{letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her



Example:
Parse Example

Transitions: SH-LA-SH-RA-SH-LA

Stack	Buffer	Arcs
$[\text{sent, her}]_S$	$[\text{letter, .}]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter



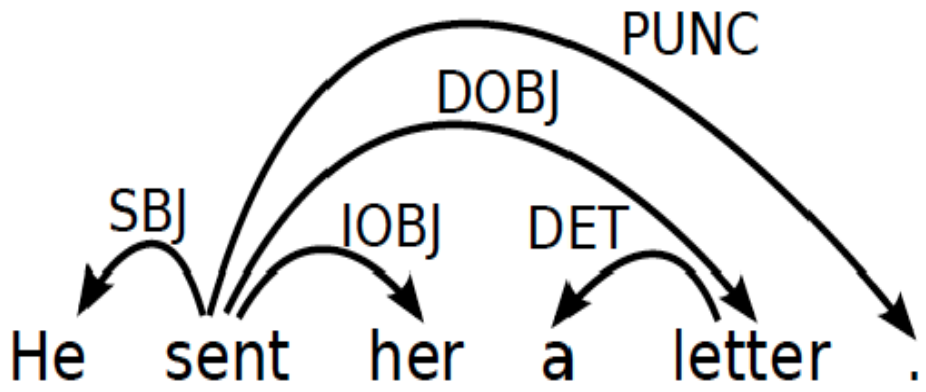
For Reduce /Shift transition:

- When there are some words in stack and some words in buffer, and referring to dependency graph, the top of stack and top of buffer are not related, then there are two possibilities, either take a reduce or a shift.
- For reduce or shift, thumb rule is, if stack has more than one word and the word just below top of stack connects with word on top of buffer then reduce otherwise shift.

Example:
Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE

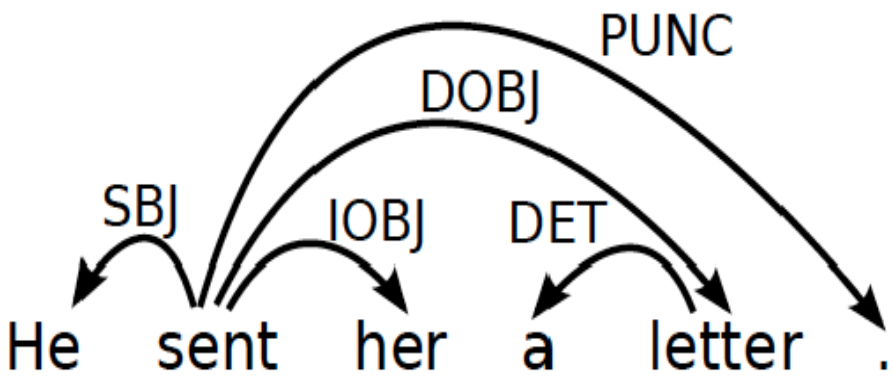
Stack	Buffer	Arcs
$[\text{sent}]_S$	$[\text{letter}, .]_B$	He $\overset{\text{SBJ}}{\leftarrow}$ sent
		sent $\overset{\text{IOBJ}}{\rightarrow}$ her
		a $\overset{\text{DET}}{\leftarrow}$ letter



Example:
Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA

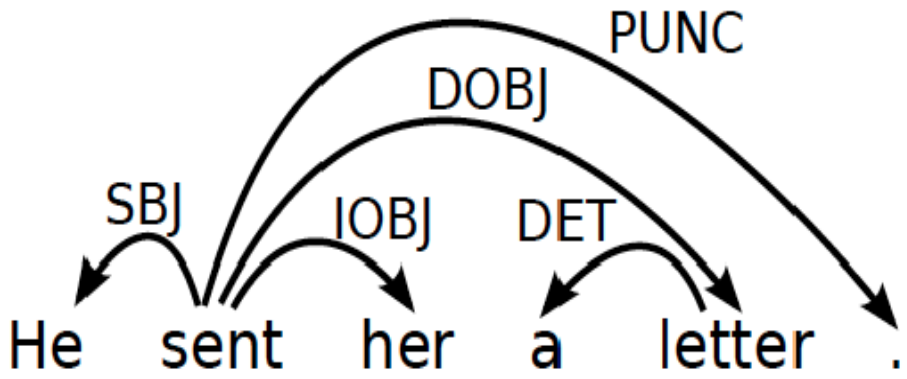
Stack	Buffer	Arcs
$[\text{sent, letter}]_S$	$[.]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter
		sent $\xrightarrow{\text{DOBJ}}$ letter



Example:
Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE

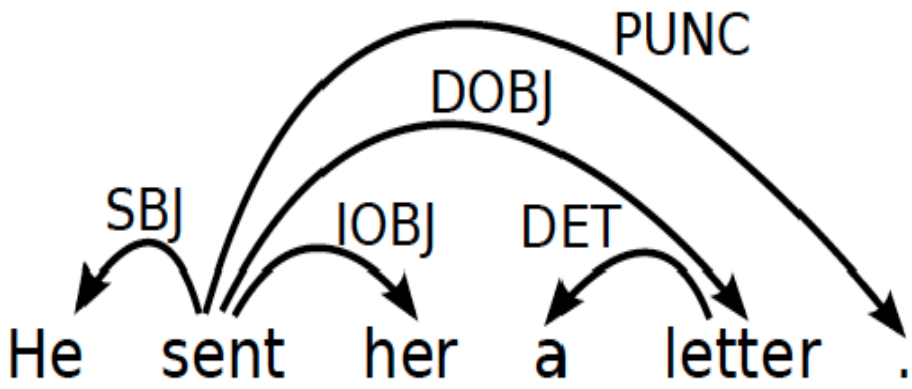
Stack	Buffer	Arcs
$[\text{sent}]_S$	$[.]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter
		sent $\xrightarrow{\text{DOBJ}}$ letter



Example:
Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

Stack	Buffer	Arcs
$[\text{sent}, .]_S$	$[\,]_B$	He $\xleftarrow{\text{SBJ}}$ sent
		sent $\xrightarrow{\text{IOBJ}}$ her
		a $\xleftarrow{\text{DET}}$ letter
		sent $\xrightarrow{\text{DOBJ}}$ letter
		sent $\xrightarrow{\text{PUNC}}$.

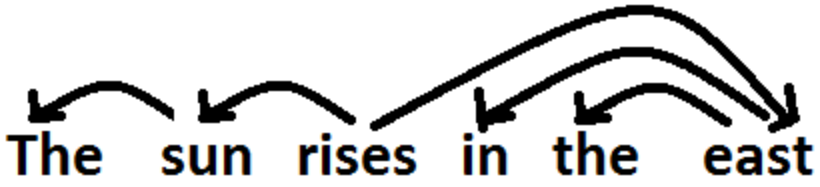


Another Example:

- Generate sequence of actions that generates the following parse tree of the sentence:

  The Sun rises in the east

  using Arc-Eager Parsing



Solution

SH-LA-SH-LA-SH-SH-SH-LA-LA-RA