Introduction to Dependency Syntax and Dependency Parsing

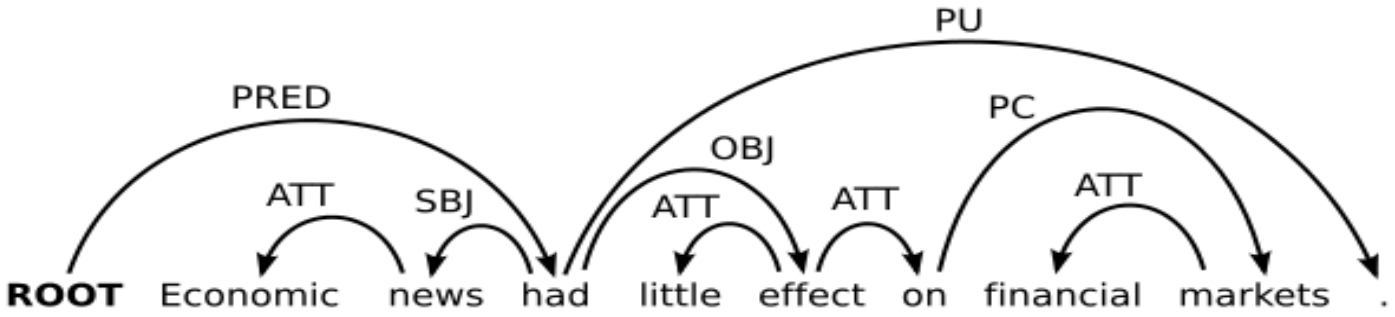
Miguel Ballesteros

Algorithms for NLP Course.
7-11

By using some materials by Joakim Nivre from Uppsala University

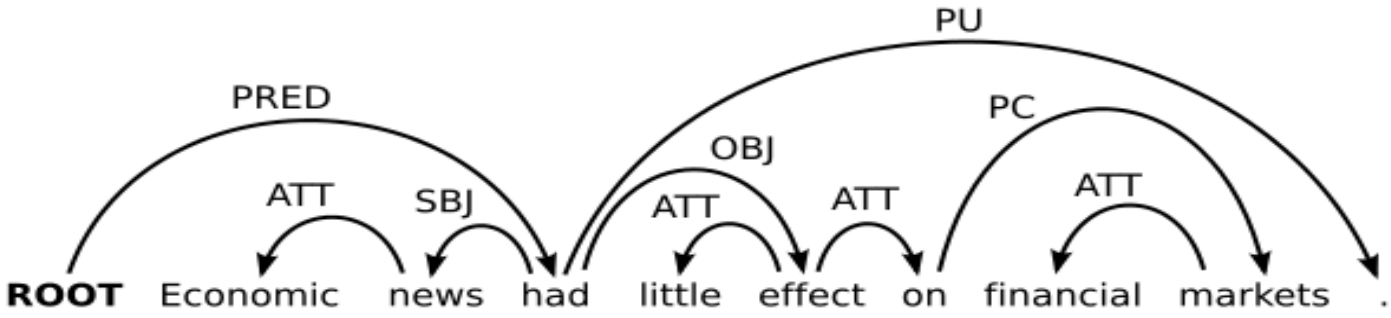
Dependency trees.

- A dependency tree is a labeled directed tree T with
  - a set V of nodes, labeled with words.
  - a set A of arcs, labeled with dependency types.
  - a linear precedence order < on V.



Dependency trees

- Labeled directed graph
  - G = (V_x,A) where
    - V_x = {0,...,n} set of nodes. One for each position of a word x_i, in the sentence plus a node 0 corresponding to a dummy ROOT node.
    - $A \subset (V_x \times L \times V_x)$ is a set of labeled arcs of the form (i,l,j), where i and j are nodes and l is the label from some set of labels L.

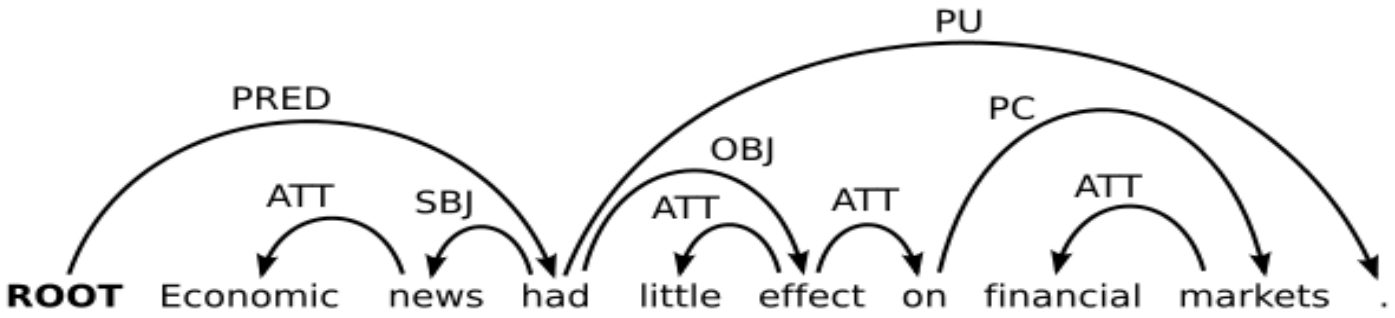


Dependency trees

- Labeled directed graph

Note that part-of-speech tags are not part of the tree (as we had in phrase-structure parsing).

In dependency parsing, we normally assume that the pos tag is an input to the parser.



Dependency trees

- The directed graph must also satisfy the following conditions:
  - **Root**: The dummy root node 0 does not have any incoming arc
    (that is, there is no arc of the form (i, l, 0)).

Dependency trees

- The directed graph must also satisfy the following conditions:
  - **Root**: The dummy root node 0 does not have any incoming arc
    (that is, there is no arc of the form (i, l, 0)).
  - **Single-Head**: Every node has at most one incoming arc (that is, the arc (i, l, j) rules out all arcs of the form (k, l , j) where k ≠ i or l' ≠ l).

Dependency trees

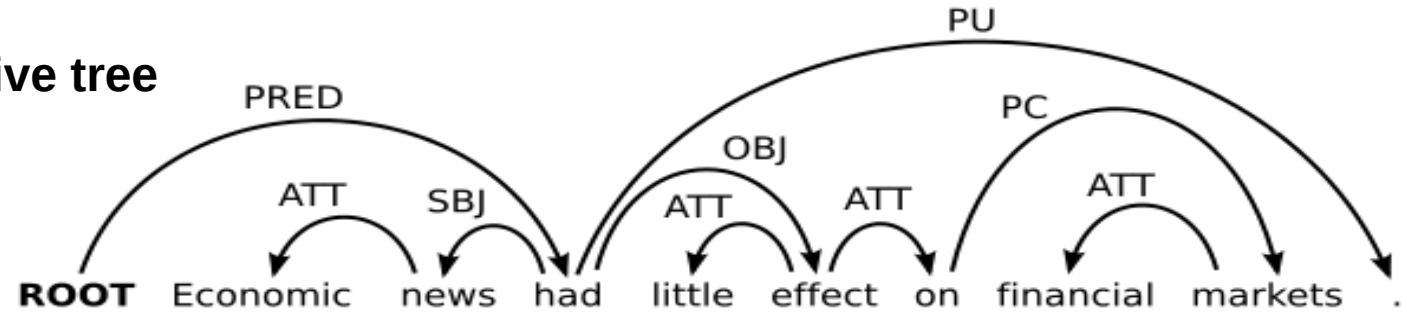
- The directed graph must also satisfy the following conditions:
  - **Root**: The dummy root node 0 does not have any incoming arc
    (that is, there is no arc of the form (i, l, 0)).
  - **Single-Head**: Every node has at most one incoming arc (that is, the arc (i, l, j) rules out all arcs of the form (k, l , j) where k ≠ i or l' ≠ l).
  - **Connected**: The graph is weakly connected (that is, in the corresponding undirected graph there is a path between any two nodes i and j).

Dependency trees

- In addition, a dependency tree may or may not satisfy the following condition:
- **Projective**: For every arc (i,l,j) there is a directed path from i to every word k such that min(i,j) < k < max(i,j)

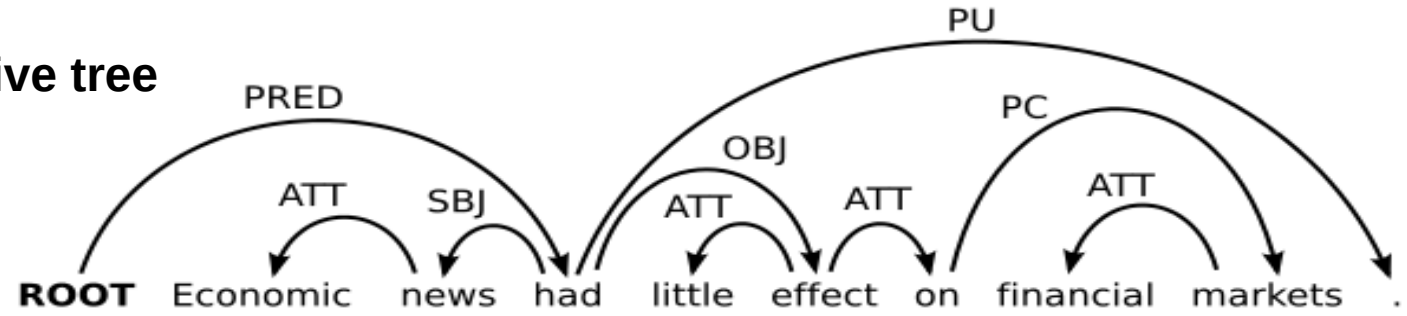
Projectivity

Projective tree

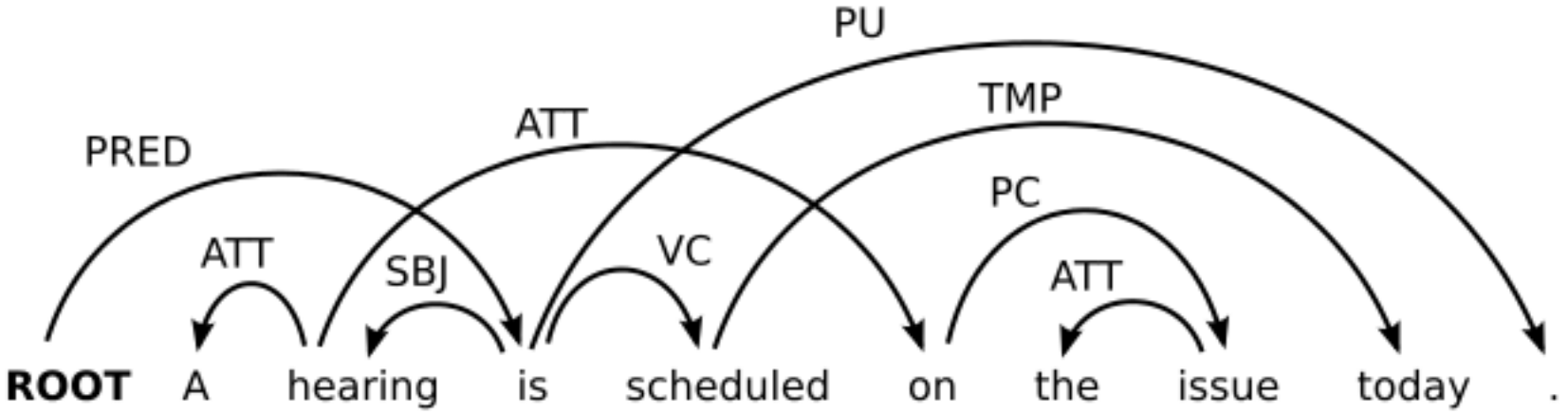


Projectivity

Projective tree



Non-projective tree



Projectivity

- **Projectivity** is a notion that has been widely discussed in the literature on dependency grammar and dependency parsing.

- Dependency grammars do not normally assume that all dependency-trees are projective, because some linguistic phenomena can only be achieved using non-projective trees.

It is heavily related to the concept of planarity in graphs.

Projectivity

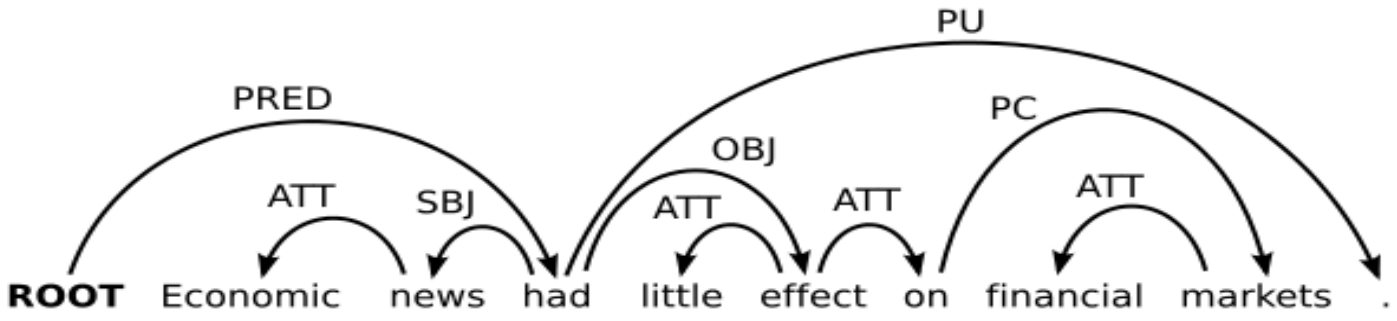
- Although, parsing non-projective trees is much harder.
- This is why a lot of parsers assume that the output trees are projective.

Detecting Projectivity/Non-Projectivity

- The idea is to use the **inorder** traversal of the tree: <left-child, root, right-child>
  - This is well defined for binary trees. We need to extend it to n-ary trees.
- If we have a projective tree, the inorder traversal will give us the original linear order.

Detecting Projectivity/Non-Projectivity

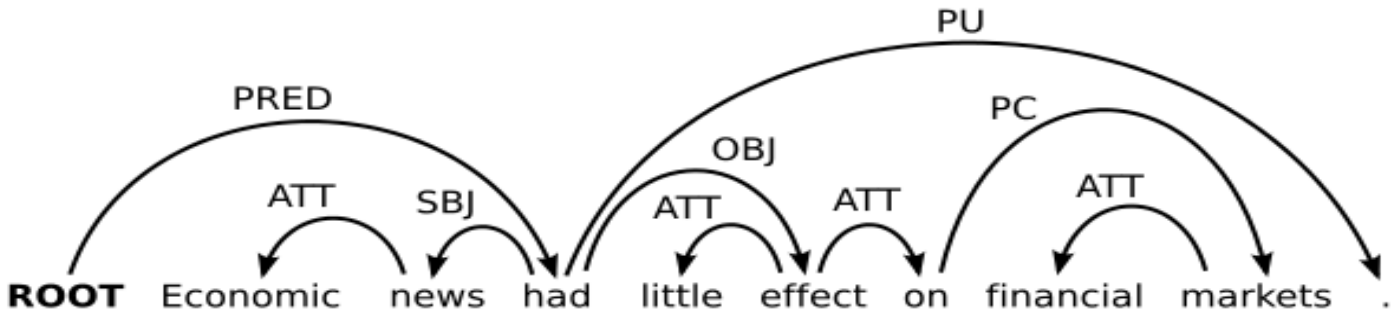
- If we have a projective tree, the inorder traversal will give us the original linear order.



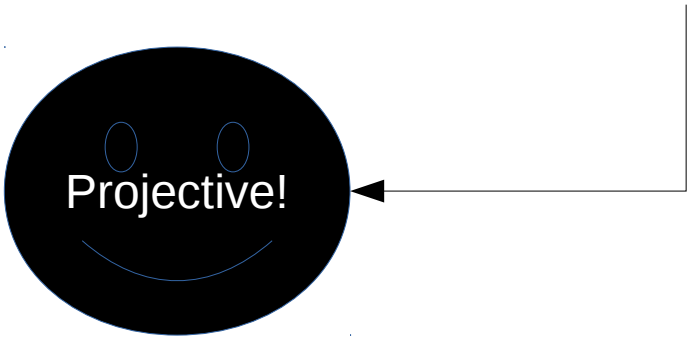
Inorder traversal: Economic news had little effect on financial markets

Detecting Projectivity/Non-Projectivity

- If we have a projective tree, the inorder traversal will give us the original linear order.

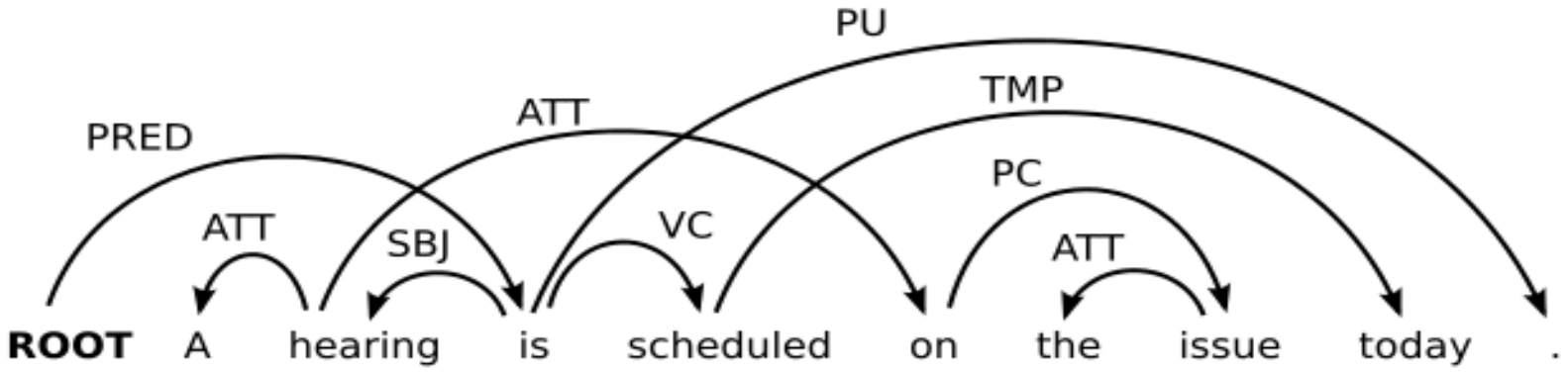


Inorder traversal: Economic news had little effect on financial markets



Detecting Projectivity/Non-Projectivity

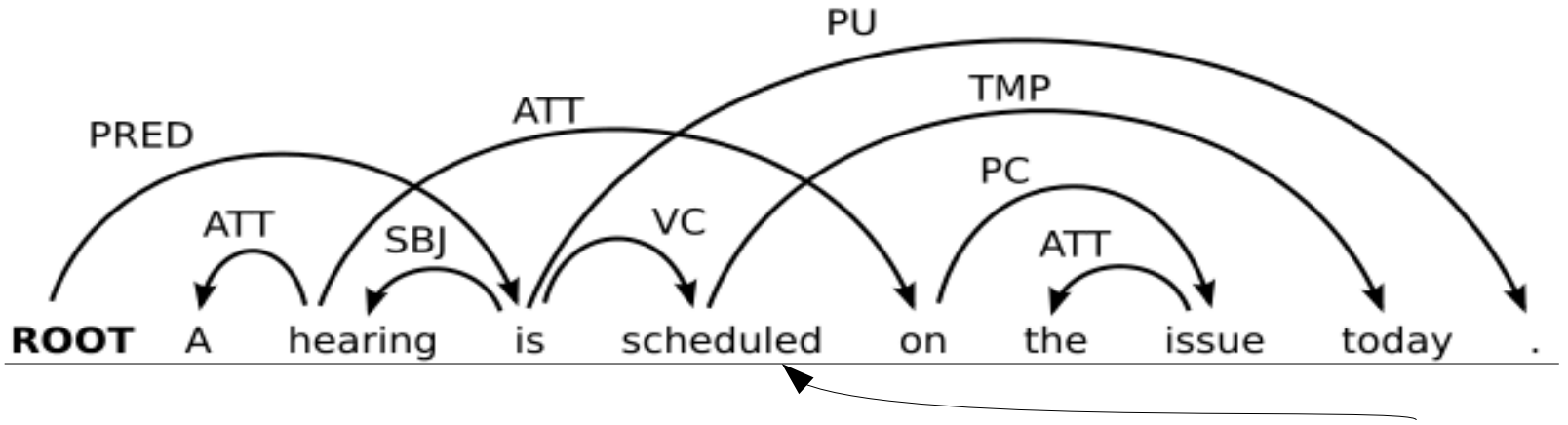
- If we have a non-projective tree, the inorder traversal will give us the original linear order.



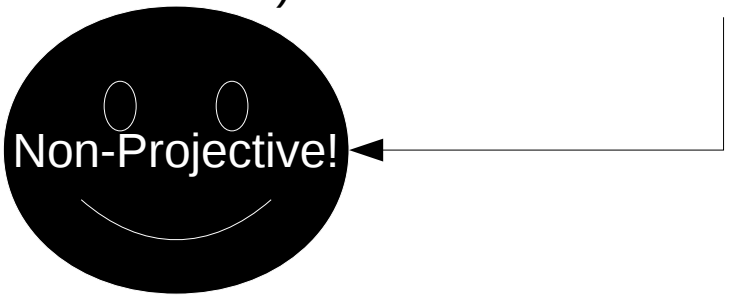
Inorder traversal: A hearing on the issue is scheduled today

Detecting Projectivity/Non-Projectivity

- If we have a non-projective tree, the inorder traversal will give us the original linear order.



Inorder traversal: Inorder traversal: A hearing on the issue is scheduled today (DOES NOT MATCH)



Non-Projective Statistics

Arabic: 11.2 %

Bulgarian: 5.4 %

Chinese: 0.0 %

Czech: 23.2 %

Danish: 15.6 %

Dutch: 36.4 %

German: 27.8 %

Japanese: 5.3 %

Polish: 18.9 %

Slovene: 22.2 %

Spanish 1.7 %

Swedish: 9.8 %

Turkish: 11.6 %

English: 0.0% (SD: 0.1%)

Percentage of non-projective trees for some treebanks of the CoNLL-X Shared Task and English.

Non-Projective Statistics

Arabic: 11.2 % (0.4% arcs)

Bulgarian: 5.4 %

Chinese: 0.0 %

Czech: 23.2 % (1.9% arcs)

Danish: 15.6 %

Dutch: 36.4 %

German: 27.8 %

Japanese: 5.3 %

Polish: 18.9 %

Slovene: 22.2 % (1.9% arcs)

Spanish 1.7 %

Swedish: 9.8 %

Turkish: 11.6 %

English: 0.0% (SD: 0.1%)

This list shows the percentage of non-projective trees: trees with at Least one Non-projective arc

Percentage of non-projective trees for some treebanks of the CoNLL-X Shared Task and English.

Parsing problem

- The parsing problem for a dependency parser is to find the optimal dependency tree *y* given an input sentence *x*.

- This amounts to assigning a syntactic head **i** and a label **l** to every node **j** corresponding to a word $\mathbf{x}_j$ in such a way that the resulting graph is a tree rooted at the node 0.

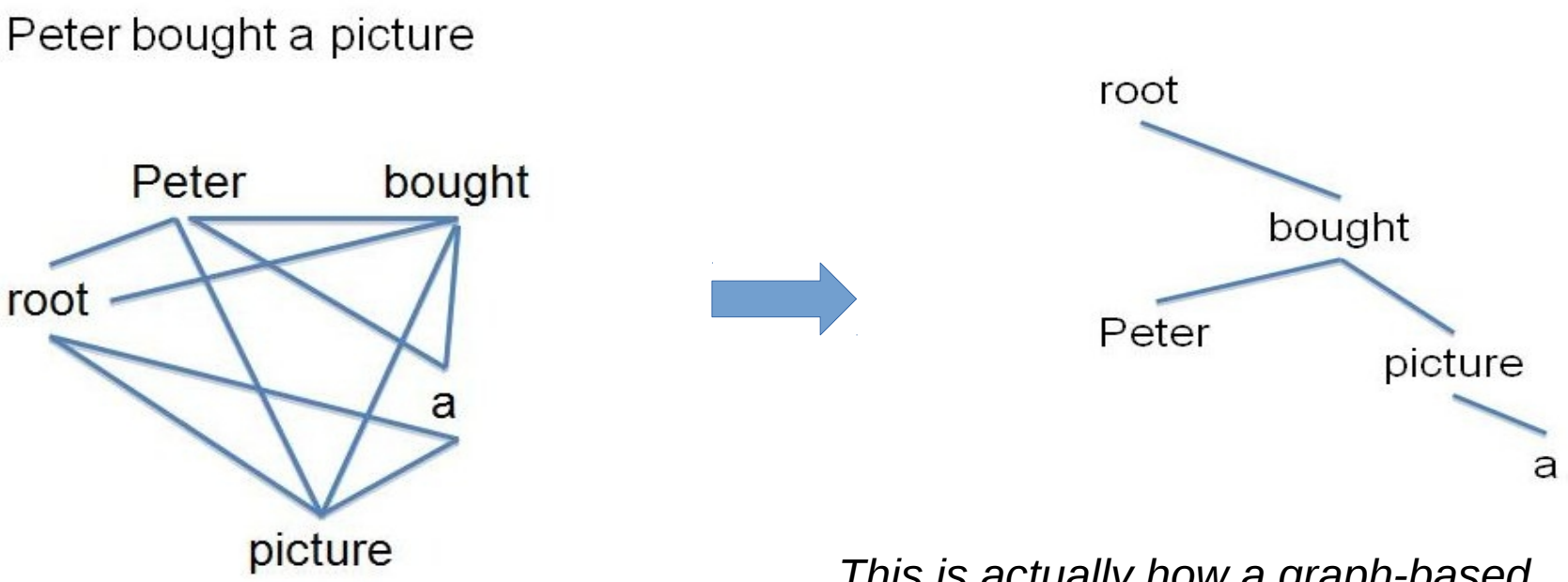
Parsing problem

Note that this Is like a tagging problem, in which each node gets a label and a head. We have the tree constraint and this makes it more complex, but each node gets a label and each node gets a head, which is another word in the input.

- This amounts to assigning a syntactic head **i** and a label **l** to every node **j** corresponding to a word $\mathbf{x}_j$ in such a way that the resulting graph is a tree rooted at the node 0.

Parsing problem

- This is equivalent to finding a spanning tree in the complete graph $G_x = (V_x , V_x \times L \times V_x)$ containing all possible arcs (i, l, j) (for nodes i, j and labels l).
- This graph G_x is also called Kn being **n** the number of nodes (Kuratowski)



This is actually how a graph-based parser works.

What about POS tags?

- A difference compared to phrase-structure parsing is that there are no part-of-speech tags in the syntactic representations.
- There are no preterminal nodes.

- Most dependency parsers assume that the input is a POS tagged sentence, though.