# Dependency Parsing

CMSC 723 / LING 723 / INST 725

MARINE CARPUAT
marine@cs.umd.edu

Slides credit: Joakim Nivre & Ryan McDonald

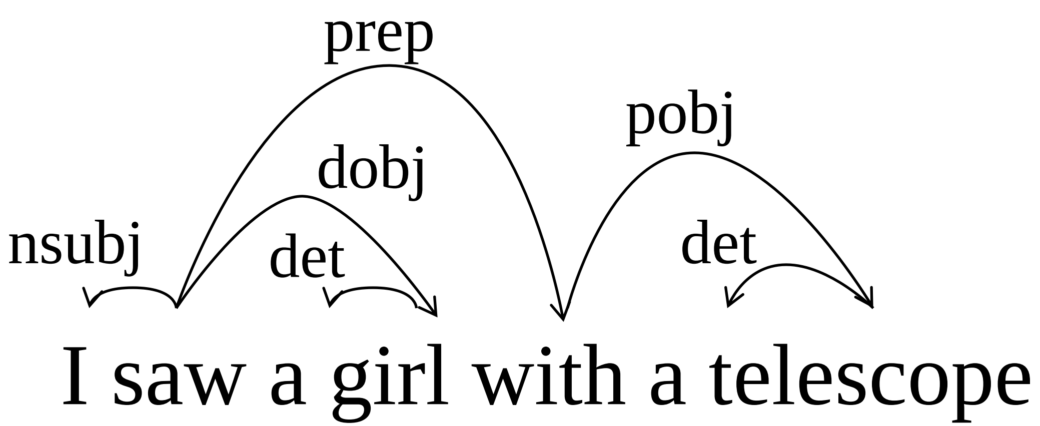
# Agenda

- Formalizing dependency graphs
- Formalizing transition-based parsing
  - Graph-based
  - Transition-based

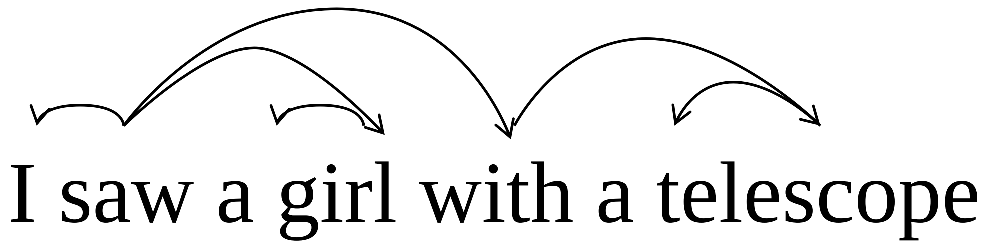
most material based on Kubler, McDonald & Nivre

# Dependencies

- Typed: Label indicating relationship between words




- Untyped: Only which words depend



# Data-driven dependency parsing

Goal: learn a good predictor of dependency graphs

Input: x

Output: dependency graph/tree G

Can be framed as a structured prediction task

- very large output space
- with interdependent labels

# FORMALIZING DEPENDENCY REPRESENTATIONS

# Dependency Graphs

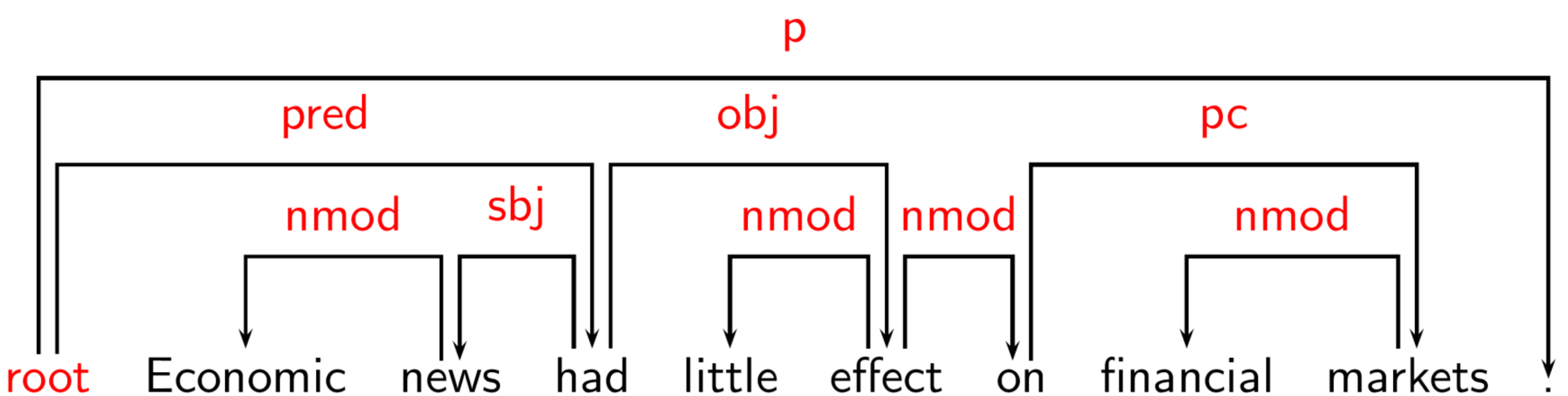
- A dependency structure can be defined as a directed graph $G$, consisting of
  - a set $V$ of nodes (vertices),
  - a set $A$ of arcs (directed edges),
  - a linear precedence order $<$ on $V$ (word order).
- Labeled graphs:
  - Nodes in $V$ are labeled with word forms (and annotation).
  - Arcs in $A$ are labeled with dependency types:
    - $L = \{l_1, \ldots, l_{|L|}\}$ 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 Graph Notation

- For a dependency graph $G = (V, A)$
- With label set $L = \{l_1, \ldots, l_{|L|}\}$
  - $i \rightarrow j \;\equiv\; \exists k : (i, j, k) \in A$
  - $i \leftrightarrow j \;\equiv\; i \rightarrow j \vee j \rightarrow i$
  - $i \rightarrow^* j \;\equiv\; i = j \vee \exists i' : i \rightarrow i',\, i' \rightarrow^* j$
  - $i \leftrightarrow^* j \;\equiv\; i = j \vee \exists i' : i \leftrightarrow i',\, i' \leftrightarrow^* j$

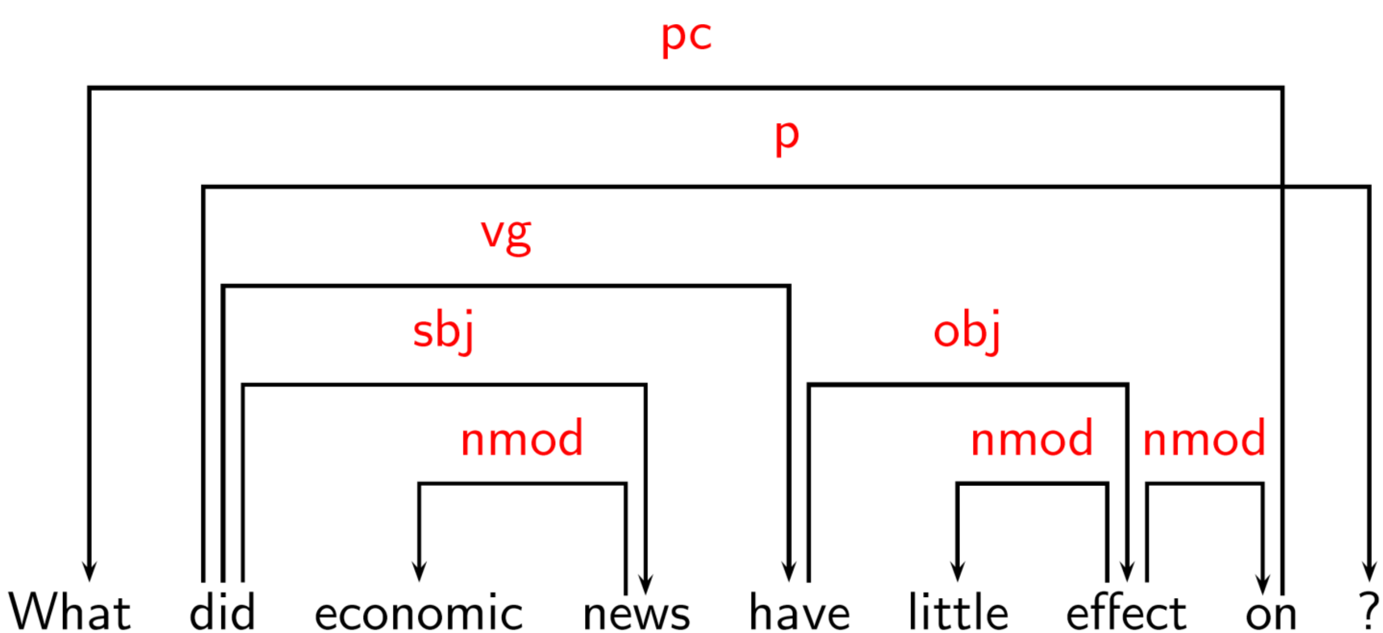
# Properties of Dependency Trees

- $G$ is (weakly) connected:
  - If $i, j \in V$, $i \leftrightarrow^* j$.
- $G$ is acyclic:
  - If $i \rightarrow j$, then not $j \rightarrow^* i$.
- $G$ obeys the single-head constraint:
  - If $i \rightarrow j$, then not $i' \rightarrow j$, for any $i' \neq i$.
- $G$ is projective:
  - If $i \rightarrow j$, then $i \rightarrow^* i'$, for any $i'$ such that $i < i' < j$ or $j < i' < i$.


p
pred
obj
pc
nmod
sbj
nmod
nmod
nmod
root Economic news had little effect on financial markets .

# Non-Projectivity

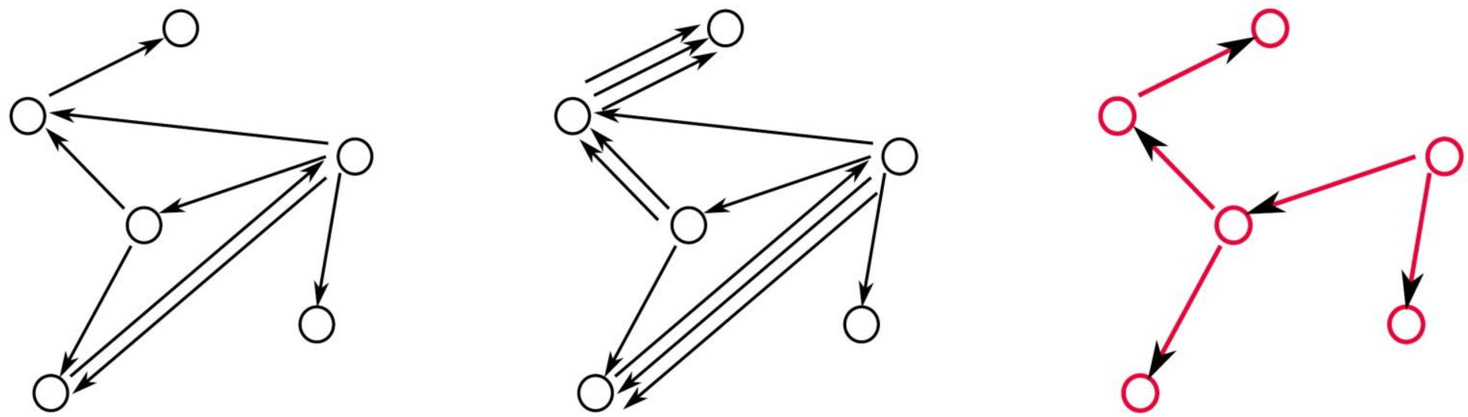
- Most theoretical frameworks do not assume projectivity
- Non-projective structures are needed to represent
  - Long-distance dependencies
  - Free word order



# GRAPH-BASED PARSING

# Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - $G'$ is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs



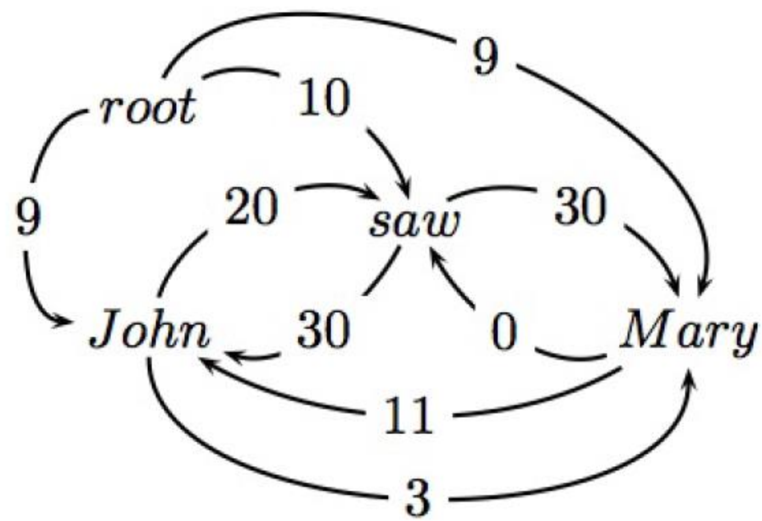
# Maximum Spanning Tree

- Assume we have an **arc factored** model

  i.e. weight of graph can be factored as sum or product of weights of its arcs

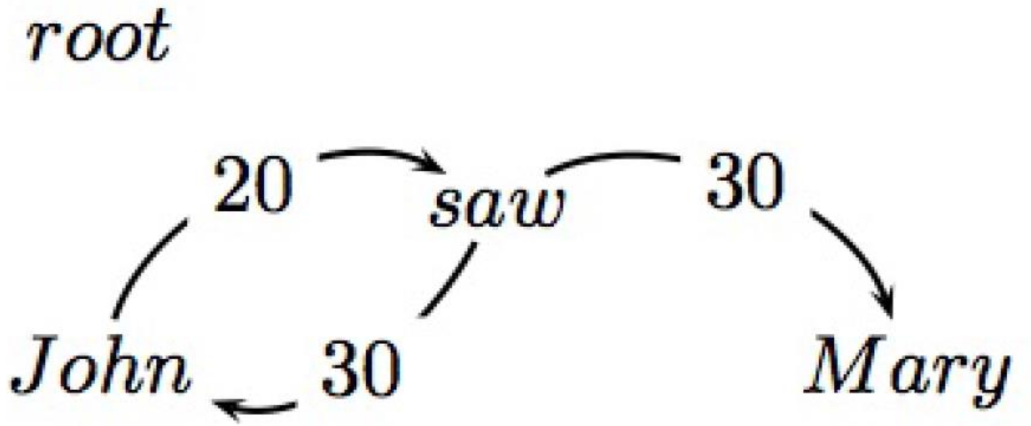
- Chu-Liu-Edmonds algorithm can find the maximum spanning tree for us!
  - Greedy recursive algorithm
  - Naïve implementation: O(n^3)

# Chu-Liu-Edmonds illustrated



# Chu-Liu-Edmonds illustrated

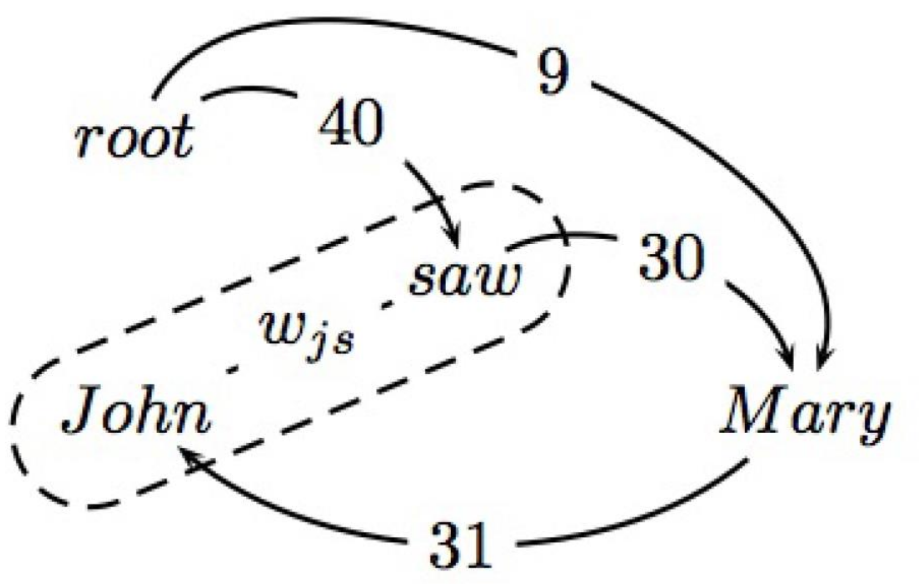
- Find highest scoring incoming arc for each vertex



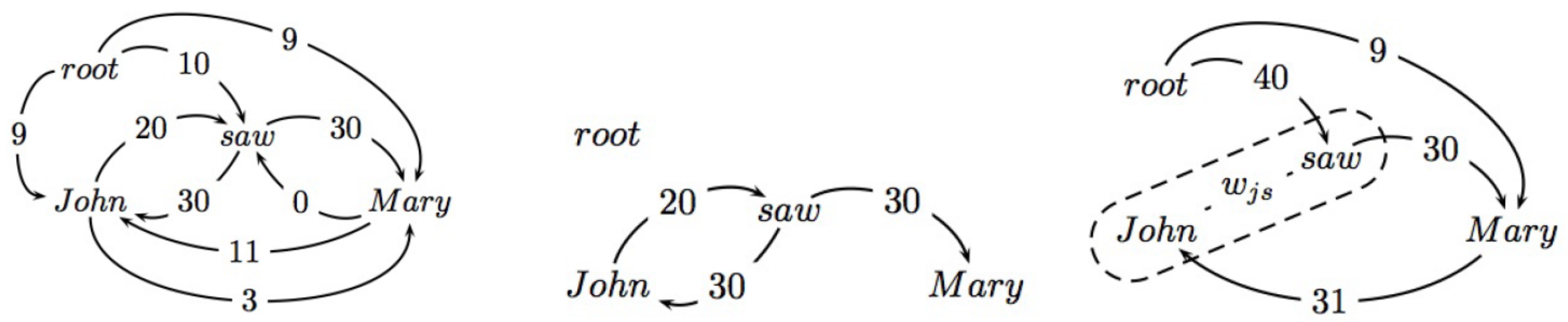

- If this is a tree, then we have found MST!!

# Chu-Liu-Edmonds illustrated

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle

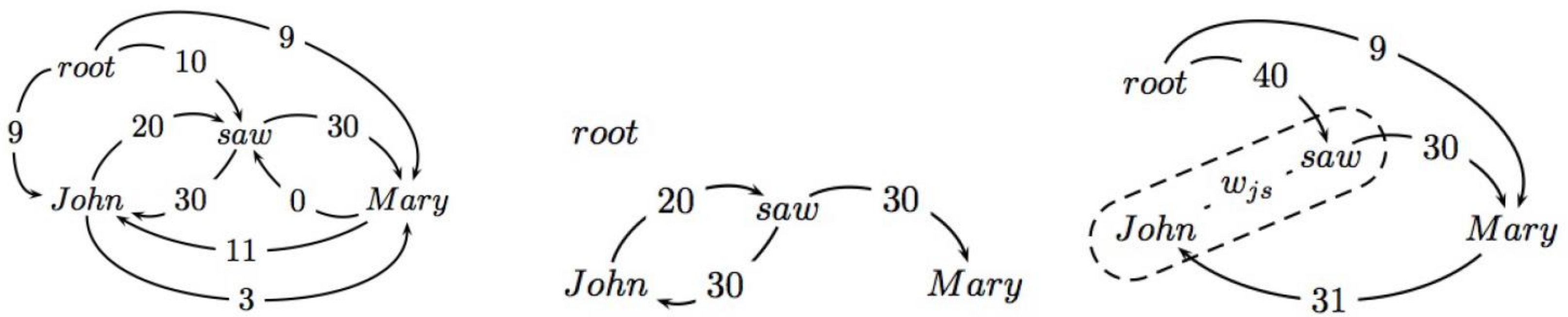


# Chu-Liu-Edmonds illustrated



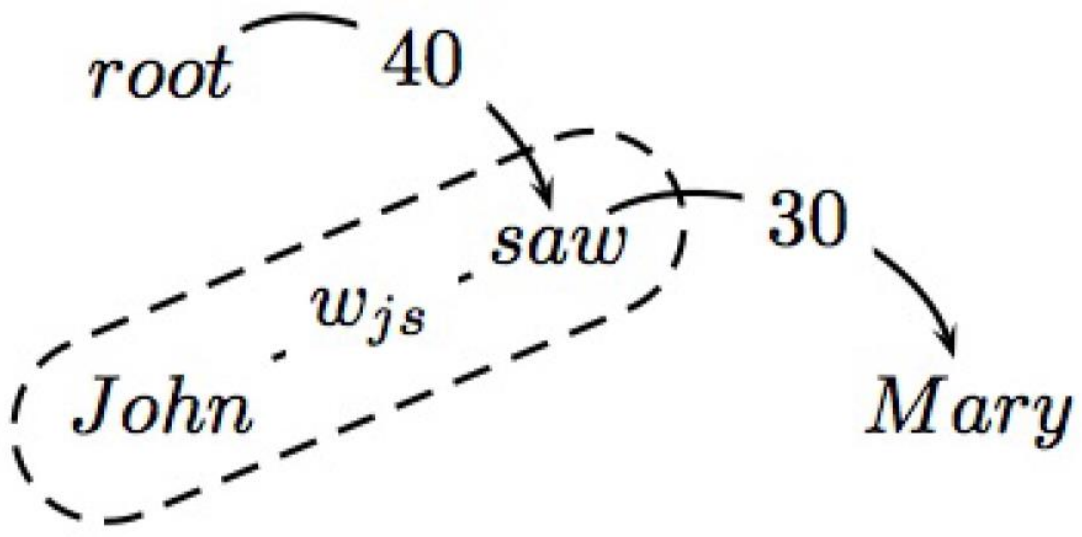

- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John $\longrightarrow$ Mary is 3 and saw $\longrightarrow$ Mary is 30

# Chu-Liu-Edmonds illustrated




- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root $\longrightarrow$ saw $\longrightarrow$ John is 40 (**)
  - root $\longrightarrow$ John $\longrightarrow$ saw is 29

- This is a tree and the MST for the contracted graph!!



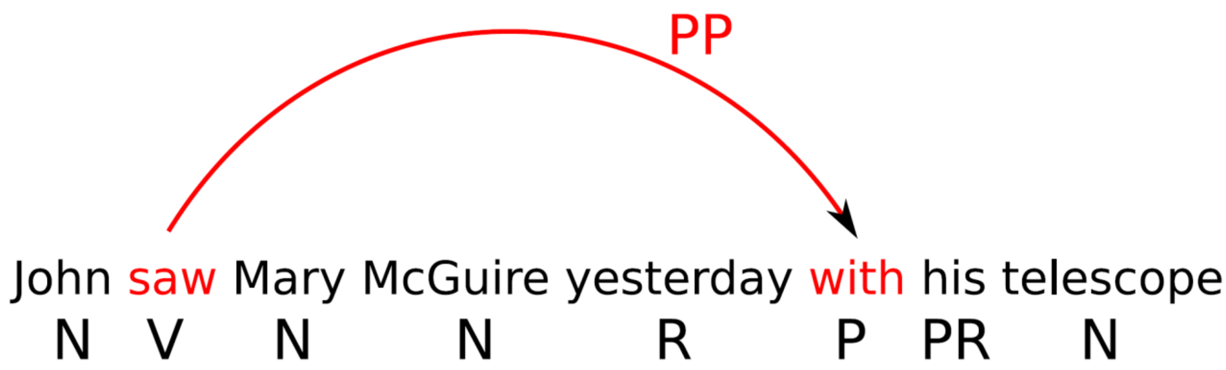

- Go back up recursive call and reconstruct final graph

# Arc weights as linear classifiers

$$w_{ij}^{k} = e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)}$$

- Arc weights are a linear combination of features of the arc, $\mathbf{f}$, and a corresponding weight vector $\mathbf{w}$
- Raised to an exponent (simplifies some math ...)
- What arc features?
- [McDonald et al. 2005] discuss a number of binary features

# Example of classifier features




- Features from [McDonald et al. 2005]:
  - Identities of the words $w_i$ and $w_j$ and the label $l_k$

head=saw & dependent=with

# How to score a graph G using features?

Arc-factored model assumption

By definition of arc weights as linear classifiers

$$
\begin{aligned}
G &= \operatorname*{arg\,max}_{G \in T(G_x)} \prod_{(i,j,k)\in G} w_{ij}^{k} = \operatorname*{arg\,max}_{G \in T(G_x)} \prod_{(i,j,k)\in G} e^{\mathbf{w}\cdot\mathbf{f}(i,j,k)} \\
&= \operatorname*{arg\,max}_{G \in T(G_x)} \log \prod_{(i,j,k)\in G} e^{\mathbf{w}\cdot\mathbf{f}(i,j,k)} \\
&= \operatorname*{arg\,max}_{G \in T(G_x)} \sum_{(i,j,k)\in G} \mathbf{w}\cdot\mathbf{f}(i,j,k) \\
&= \operatorname*{arg\,max}_{G \in T(G_x)} \mathbf{w}\cdot \sum_{(i,j,k)\in G} \mathbf{f}(i,j,k) = \operatorname*{arg\,max}_{G \in T(G_x)} \mathbf{w}\cdot\mathbf{f}(G)
\end{aligned}
$$

# How can we learn the classifier from data?

e.g., The Perceptron

Training data: $\mathcal{T} = \{(x_t, G_t)\}_{t=1}^{|\mathcal{T}|}$

$\mathbf{w}^{(0)} = 0;\ i = 0$

for $n : 1..N$

  for $t : 1..T$

    Let $G' = \arg\max_{G'} \mathbf{w}^{(i)} \cdot \mathbf{f}(G')$

    if $G' \neq G_t$

      $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{f}(G_t) - \mathbf{f}(G')$

      $i = i + 1$

return $\mathbf{w}^{i}$

# TRANSITION-BASED DEPENDENCY PARSER

# Transition-based parsing

- A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, where
  1. $C$ is a set of configurations, each of which contains a buffer $\beta$ of (remaining) nodes and a set $A$ of dependency arcs,
  2. $T$ is a set of transitions, each of which is a (partial) function $t : C \to C$,
  3. $c_s$ is an initialization function, mapping a sentence $x = w_0, w_1, \ldots, w_n$ to a configuration with $\beta = [1, \ldots, n]$,
  4. $C_t \subseteq C$ is a set of terminal configurations.
- Note:
  - A configuration represents a parser state.
  - A transition represents a parsing action (parser state update).

# Transition-based parsing

- Let $S = (C, T, c_s, C_t)$ be a transition system.
- A transition sequence for a sentence $x = w_0, w_1, \ldots, w_n$ in $S$ is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that
  1. $c_0 = c_s(x)$,
  2. $c_m \in C_t$,
  3. for every $i$ $(1 \leq i \leq m)$, $c_i = t(c_{i-1})$ for some $t \in T$.
- The parse assigned to $x$ by $C_{0,m}$ is the dependency graph $G_{c_m} = (\{0, 1, \ldots, n\}, A_{c_m})$, where $A_{c_m}$ is the set of dependency arcs in $c_m$.

# Deterministic parsing with an oracle

- An oracle for a transition system $S = (C, T, c_s, C_t)$ is a function $o : C \to T$.
- Given a transition system $S = (C, T, c_s, C_t)$ and an oracle $o$, deterministic parsing can be achieved by the following simple algorithm:

Parse($x = (w_0, w_1, \ldots, w_n)$)

$c \leftarrow c_s(x)$

**while** $c \notin C_t$

$\quad c = [o(c)](c)$

**return** $G_c$

# Stack-based transition system

- A stack-based configuration for a sentence $x = w_0, w_1, \ldots, w_n$ is a triple $c = (\sigma, \beta, A)$, where
  1. $\sigma$ is a stack of tokens $i \leq m$ (for some $m \leq n$),
  2. $\beta$ is a buffer of tokens $j > m$,
  3. $A$ is a set of dependency arcs such that $G = (\{0, 1, \ldots, n\}, A)$ is a dependency graph for $x$.
- A stack-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where
  1. $C$ is the set of all stack-based configurations,
  2. $c_s(x = w_0, w_1, \ldots w_n) = ([0], [1, \ldots, n], \emptyset)$,
  3. $T$ is a set of transitions, each of which is a function $t : C \rightarrow C$,
  4. $C_t = \{c \in C | c = (\sigma, [\,], A)\}$.

# Transitions & Preconditions

- Transitions:
  - Left-Arc$_k$:

    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$

  - Right-Arc$_k$:

    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j, k)\})$$

  - Shift:

    $$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$$

- Preconditions:
  - Left-Arc$_k$:

    $$\neg[i = 0]$$
    $$\neg\exists i' \exists k' [(i', i, k') \in A]$$

  - Right-Arc$_k$:

    $$\neg\exists i' \exists k' [(i', j, k') \in A]$$

# Let's try it out...

- Transitions:
  - Left-Arc$_k$:

    $(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$
  - Right-Arc$_k$:

    $(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j, k)\})$
  - Shift:

    $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
- Preconditions:
  - Left-Arc$_k$:

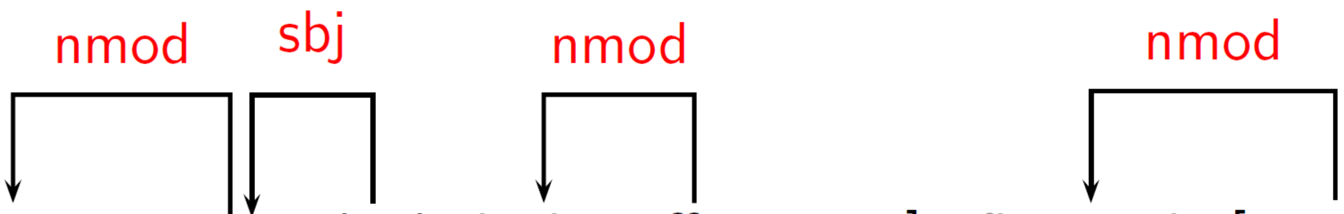
    $\neg[i = 0]$

    $\neg\exists i' \exists k' [(i', i, k') \in A]$
  - Right-Arc$_k$:

    $\neg\exists i' \exists k' [(i', j, k') \in A]$

$[\text{root}_0]_\sigma$ $[\text{Economic}_1$ $\text{news}_2$ $\text{had}_3$ $\text{little}_4$ $\text{effect}_5$ $\text{on}_6$ $\text{financial}_7$ $\text{markets}_8$ $._9]_\beta$

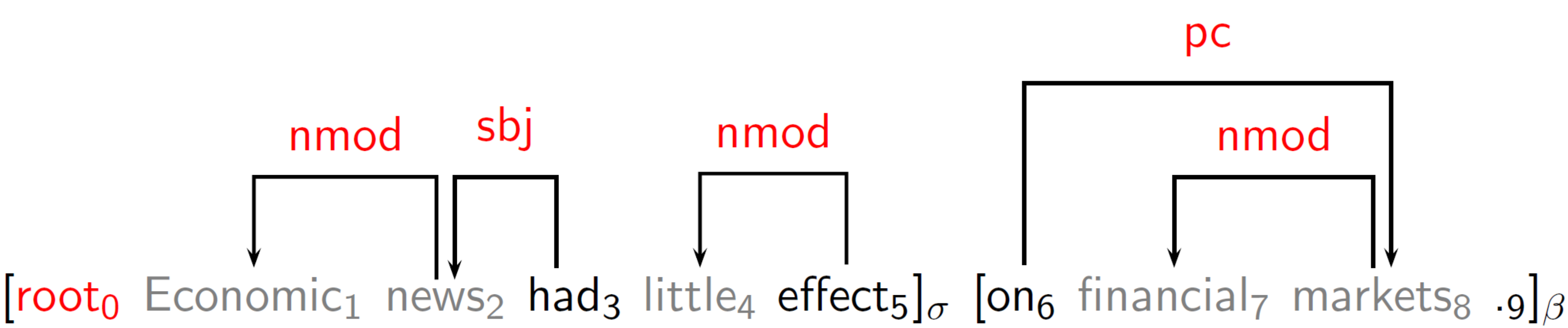
# A few steps illustrated...




[root$_0$ Economic$_1$ news$_2$ had$_3$ little$_4$ effect$_5$ on$_6$]$_\sigma$ financial$_7$ [markets$_8$ .$_9$]$_\beta$

Left-Arc$_{nmod}$

# A few steps illustrated...




Right-Arc$_{pc}$