COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Dependency Parsing II

CMSC 470

Marine Carpuat

Arc Standard Transition System defines 3 transition operators [Covington, 2001; Nivre 2003]

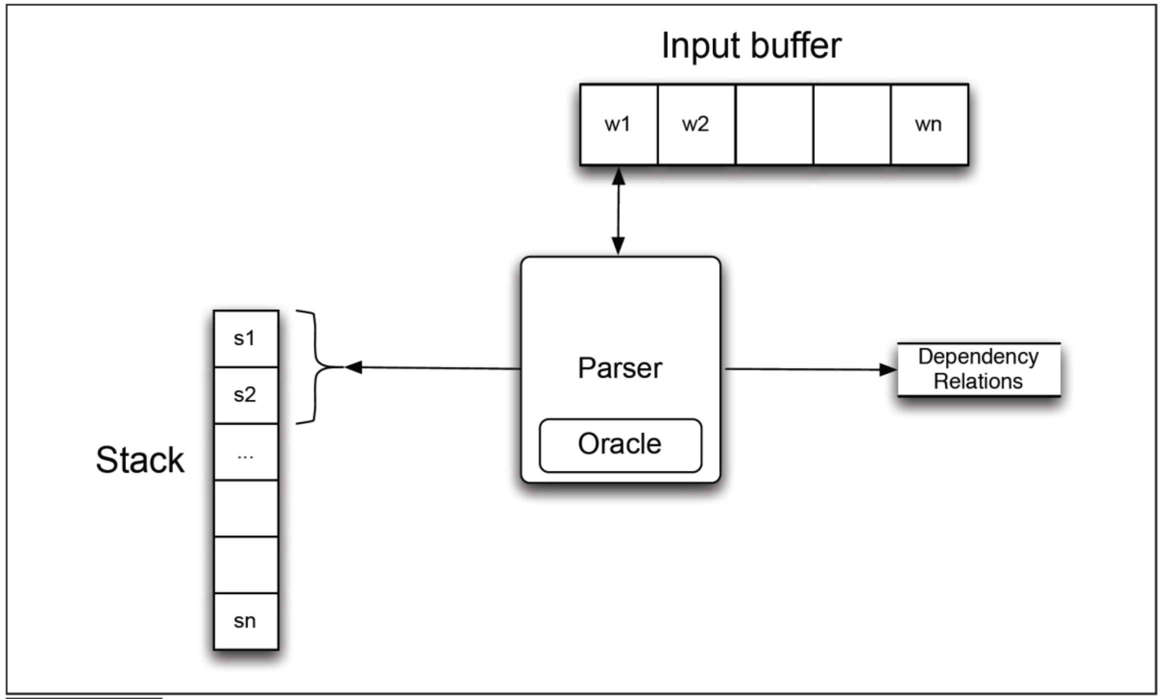


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

SHIFT

- Remove word at head of input buffer
- Push it on the stack

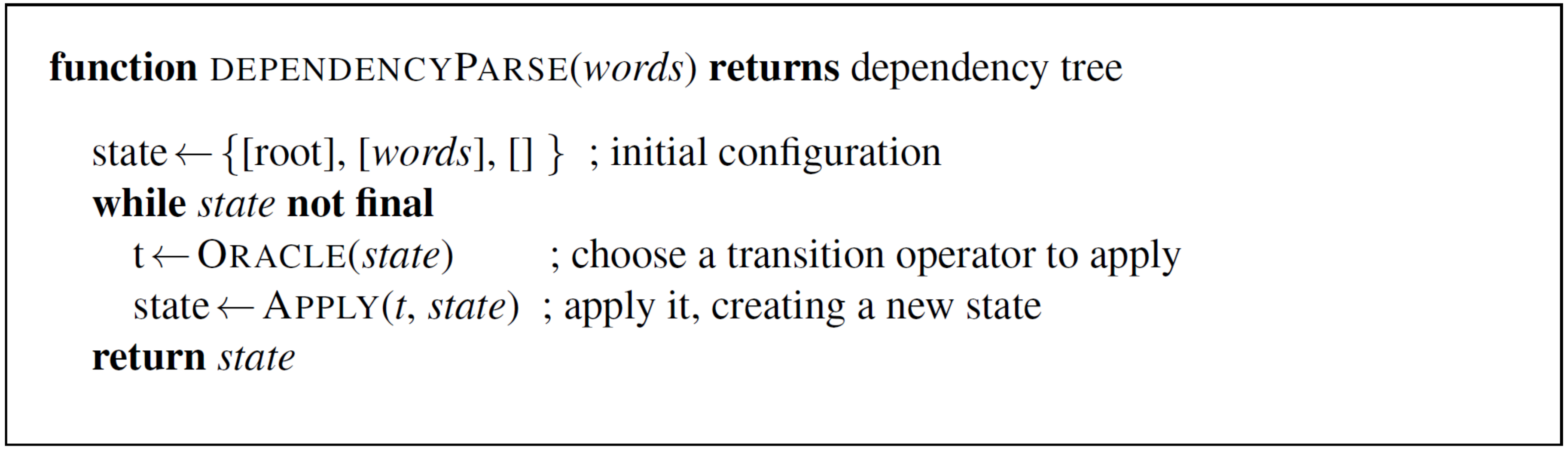
LEFT-ARC

- create head-dependent relation between word at top of stack and 2nd word (under top)
- remove 2nd word from stack

RIGHT-ARC

- Create head-dependent relation between word on 2nd word on stack and word on top
- Remove word at top of stack

Transition-based Dependency Parser



```
function DEPENDENCYPARSE(words) returns dependency tree

    state ← {[root], [words], [] }  ; initial configuration
    while state not final
        t ← ORACLE(state)          ; choose a transition operator to apply
        state ← APPLY(t, state)  ; apply it, creating a new state
    return state
```

Figure 14.6 A generic transition-based dependency parser

Properties of this algorithm:
- Linear in sentence length
- A greedy algorithm
- Output quality depends on oracle

Research highlight: Dependency parsing with stack-LSTMs

- From Dyer et al. 2015: http://www.aclweb.org/anthology/P15-1033

- Idea
  - Instead of hand-crafted feature
  - Predict next transition using recurrent neural networks to learn representation of stack, buffer, sequence of transitions

Research highlight: Dependency parsing with stack-LSTMs

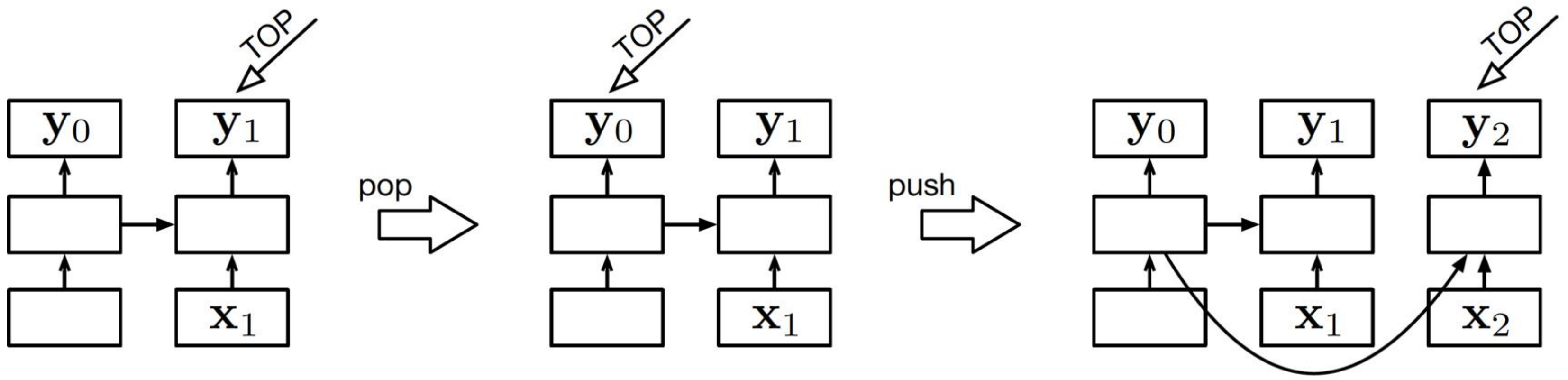


Figure 1: A stack LSTM extends a conventional left-to-right LSTM with the addition of a stack pointer (notated as TOP in the figure). This figure shows three configurations: a stack with a single element (left), the result of a pop operation to this (middle), and then the result of applying a push operation (right). The boxes in the lowest rows represent stack contents, which are the inputs to the LSTM, the upper rows are the outputs of the LSTM (in this paper, only the output pointed to by TOP is ever accessed), and the middle rows are the memory cells (the $\mathbf{c}_t$'s and $\mathbf{h}_t$'s) and gates. Arrows represent function applications (usually affine transformations followed by a nonlinearity), refer to §2.1 for specifics.

Research highlight: Dependency parsing with stack-LSTMs

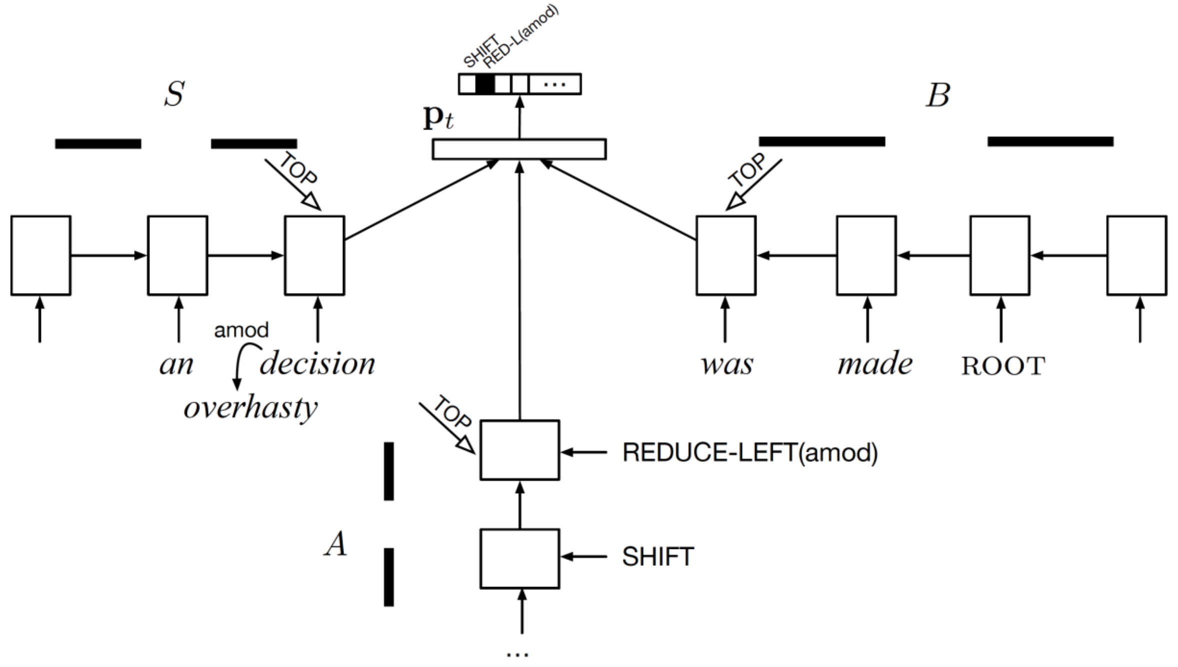
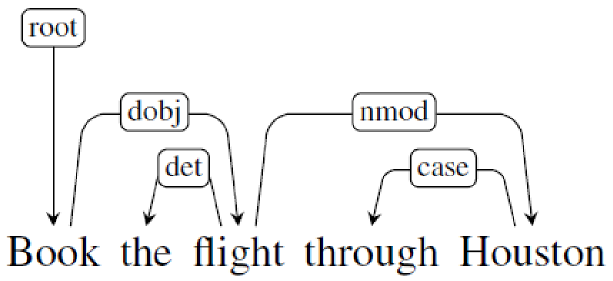


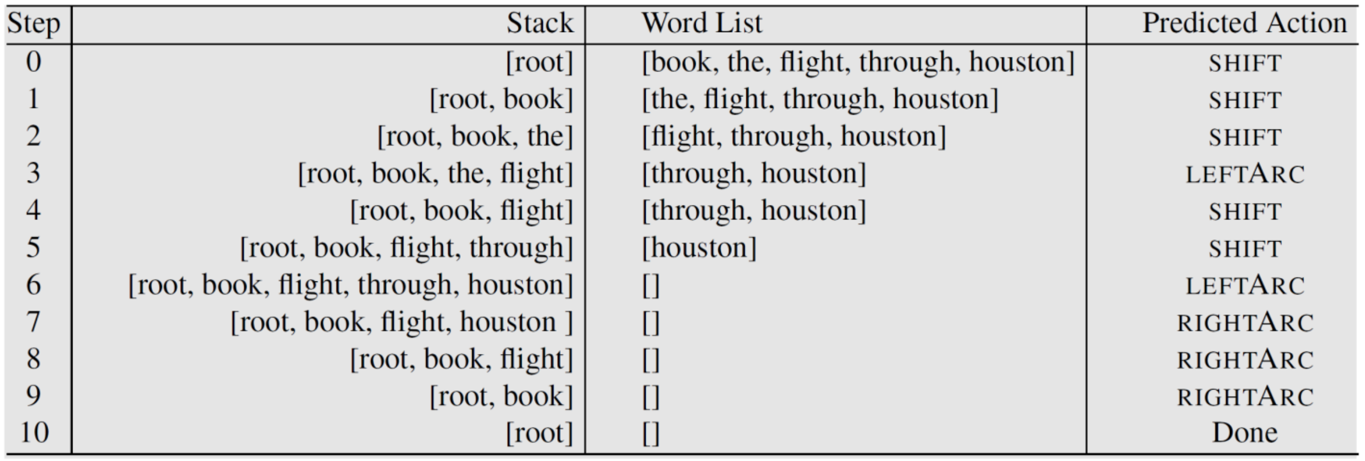
Figure 2: Parser state computation encountered while parsing the sentence "*an overhasty decision was made*." Here S designates the stack of partially constructed dependency subtrees and its LSTM encoding; B is the buffer of words remaining to be processed and its LSTM encoding; and A is the stack representing the history of actions taken by the parser. These are linearly transformed, passed through a ReLU nonlinearity to produce the parser state embedding $\mathbf{p}_t$. An affine transformation of this embedding is passed to a softmax layer to give a distribution over parsing decisions that can be taken.

An Alternative to the Arc-Standard Transition System

A weakness of arc-standard parsing

Right dependents cannot be attached to their head until all their dependents have been attached





Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 14.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

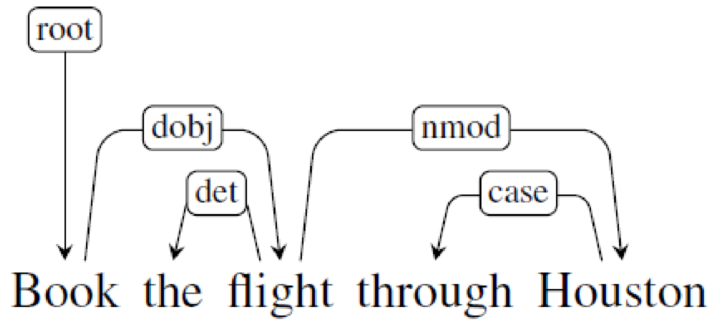
Arc Eager Parsing

- LEFT-ARC
  - Create head-dependent rel. between word at front of buffer and word at top of stack
  - pop the stack
- RIGHT-ARC
  - Create head-dependent rel. between word on top of stack and word at front of buffer
  - Shift buffer head to stack
- SHIFT
  - Remove word at head of input buffer
  - Push it on the stack
- REDUCE
  - Pop the stack

Move dependent word to stack (so it can serve as head of other words)

Pop words off the stack once they have been assigned all their dependents

Arc Eager Parsing Example



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	

Figure 14.10 A processing trace of *Book the flight through Houston* using the arc-eager transition operators.

Properties of transition-based parsing algorithms

Trees & Forests

- A dependency tree is a graph satisfying the following conditions
  - Root
  - Single head
  - No cycles
  - Connectedness

- A dependency forest is a dependency graph satisfying
  - Root
  - Single head
  - No cycles
  - but **not** Connectedness

Properties of the transition-based parsing algorithm we've seen

Soundness: For every complete transition sequence, the resulting graph is a *projective* dependency forest

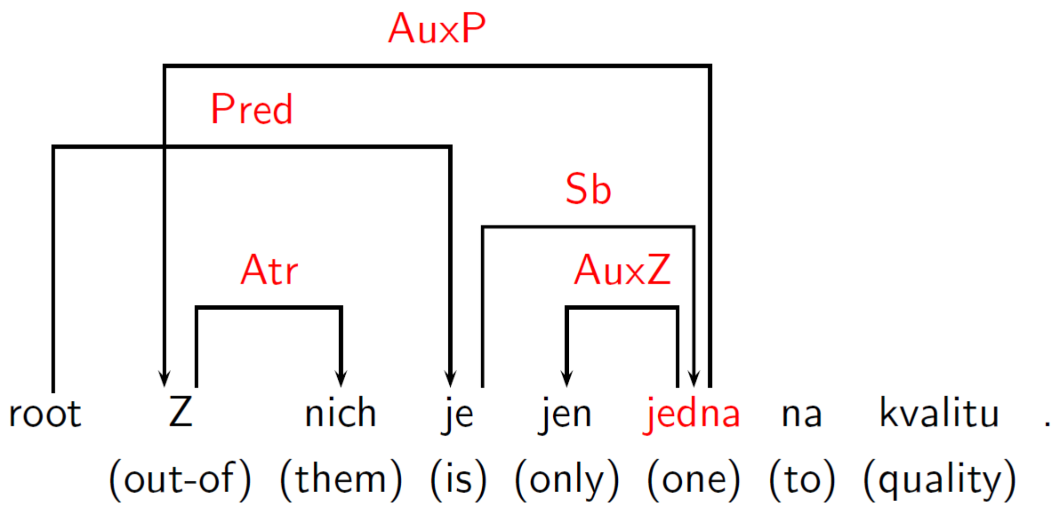
Completeness: For every *projective* dependency forest G, there is a transition sequence that generates G

If we really want a tree rather than a forest, we can use a trick: add links to ROOT from disconnected trees

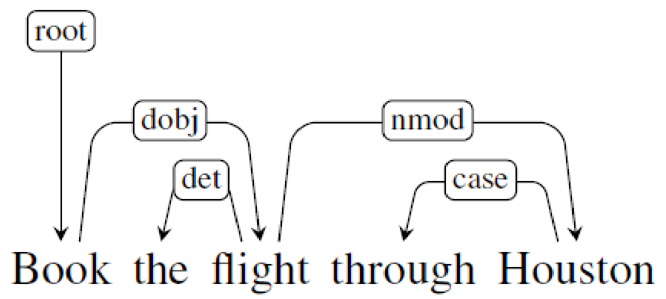
Projectivity

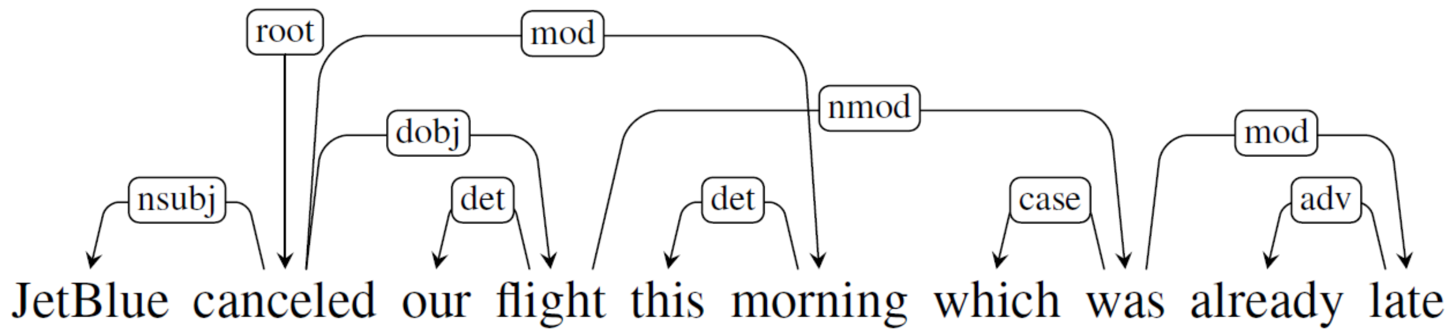
- **Arc** from head to dependent is **projective**
  - If there is a path from head to every word between head and dependent
- **Dependency tree** is **projective**
  - If all arcs are projective
  - Or equivalently, if it can be drawn with no crossing edges

Is this tree projective?



Is this tree projective?




root
mod
nmod
dobj
mod
nsubj
det
det
case
adv
JetBlue canceled our flight this morning which was already late

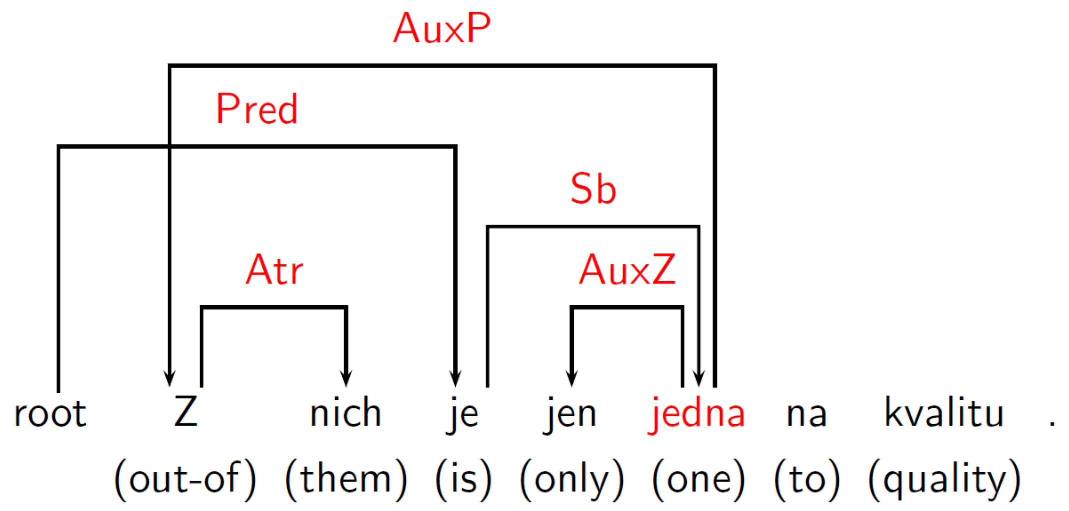
Projectivity

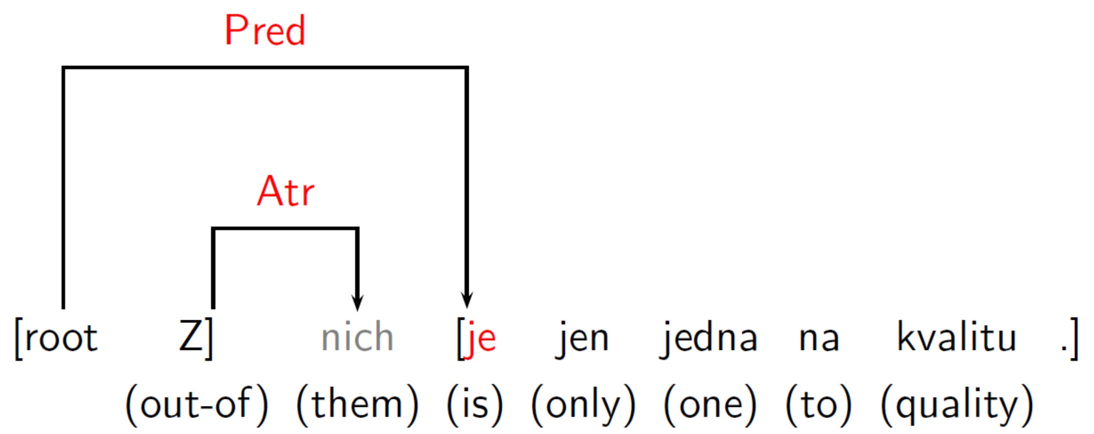
- **Arc** from head to dependent is **projective**
  - If there is a path from head to every word between head and dependent

- **Dependency tree** is **projective**
  - If all arcs are projective
  - Or equivalently, if it can be drawn with no crossing edges

- Projective trees make computation easier
- But most theoretical frameworks do not assume projectivity
  - Need to capture long-distance dependencies, free word order

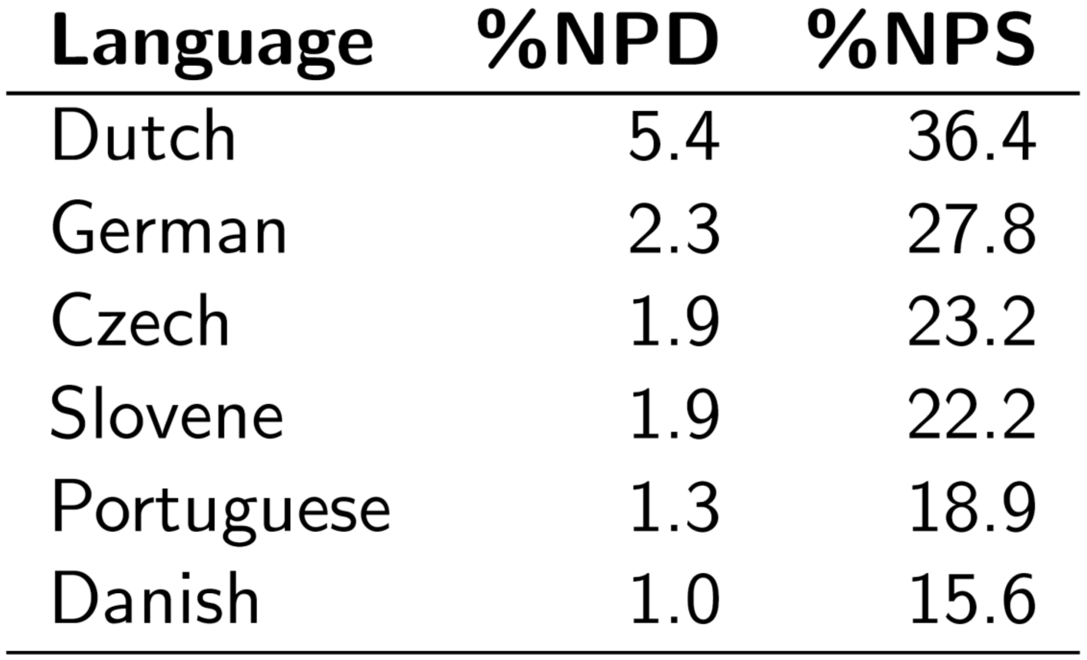
Arc-standard parsing can’t produce non-projective trees




Pred
Atr
[root
Z]
nich
[je
jen
jedna
na
kvalitu
.]
(out-of)
(them)
(is)
(only)
(one)
(to)
(quality)

How frequent are non-projective structures?

- Statistics from CoNLL shared task
  - NPD = non projective dependencies
  - NPS = non projective sentences



Language	**%NPD**	**%NPS**
Dutch	5.4	36.4
German	2.3	27.8
Czech	1.9	23.2
Slovene	1.9	22.2
Portuguese	1.3	18.9
Danish	1.0	15.6

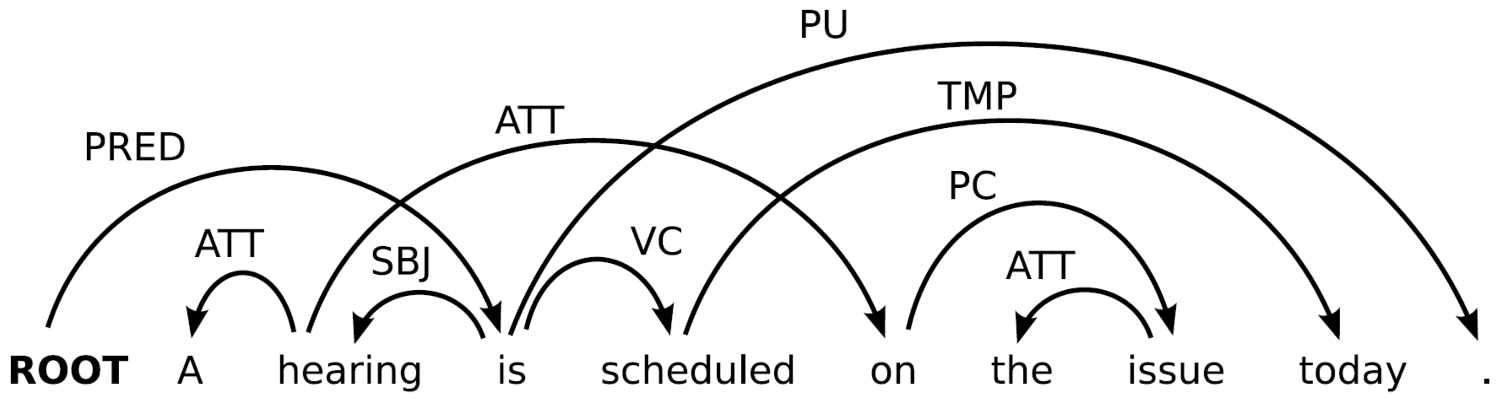
How to deal with non-projectivity? (1) change the transition system

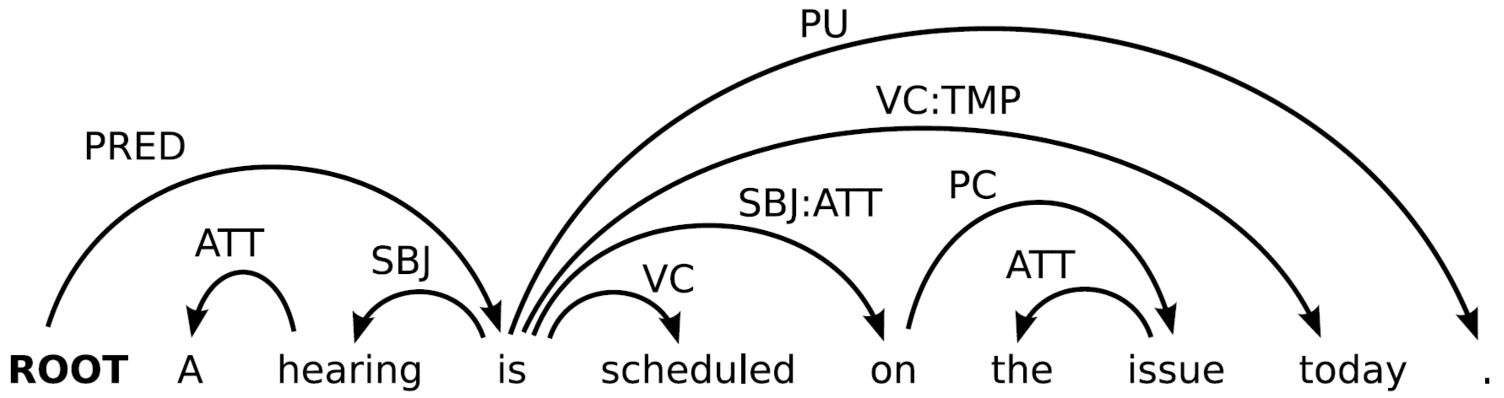
Transition		Preconditio
NP-Left$_r$	$(\sigma\|w_i\|w_k, w_j\|\beta, A) \Rightarrow (\sigma\|w_k, w_j\|\beta, A \cup \{(w_j, r, w_i)\})$	$i \neq 0$
NP-Right$_r$	$(\sigma\|w_i\|w_k, w_j\|\beta, A) \Rightarrow (\sigma\|w_i, w_k\|\beta, A \cup \{(w_i, r, w_j)\})$	

- Intuition
  - Add new transitions
    - That apply to 2nd word of the stack
    - Top word of stack is treated as context

[Attardi 2006]

How to deal with non-projectivity? (2) pseudo-projective parsing





Intuition

- "projectivize" a non-projective tree
- by creating new projective arcs that can be transformed back into non-projective arcs in a post-processing step

Dependency Parsing: what you should know

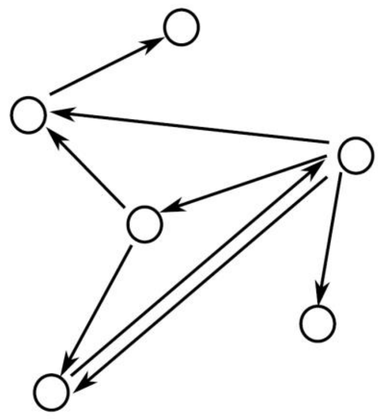
- Transition-based dependency parsing
  - Shift-reduce parsing
  - Transition systems: arc standard, arc eager
  - Oracle algorithm: how to obtain a transition sequence given a tree
  - How to construct a multiclass classifier to predict parsing actions
  - What transition-based parsers can and cannot do
  - That transition-based parsers provide a flexible framework that allows many extensions
    - such as RNNs vs feature engineering, non-projectivity (but I don’t expect you to memorize these algorithms)
- Next: Graph-based dependency parsing

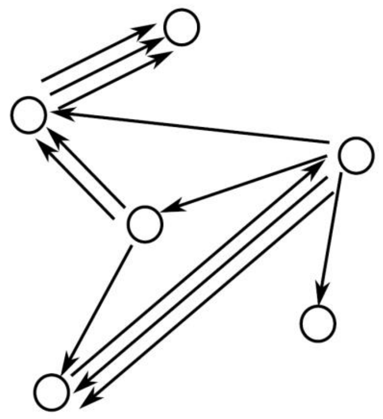
Graph-based Dependency Parsing

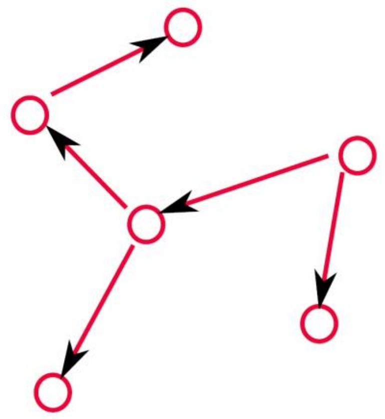
Slides credit: Joakim Nivre

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







Dependency Parsing as Finding the Maximum Spanning Tree

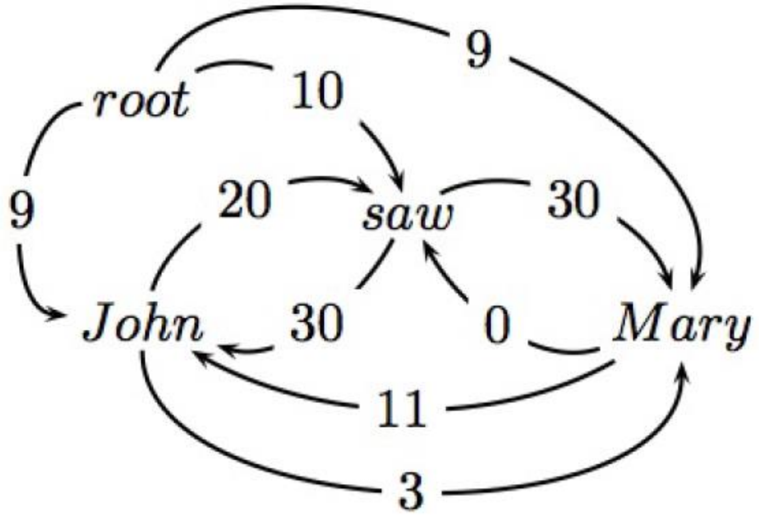
- Views parsing as finding the best directed spanning tree
  - of multi-digraph that captures all possible dependencies in a sentence
  - needs a score that quantifies how good a tree is

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

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

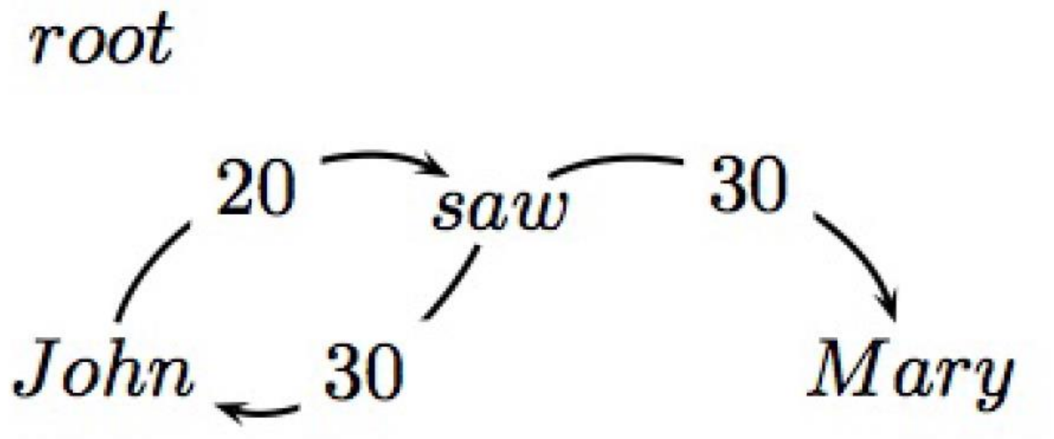
- Chu-Liu-Edmonds algorithm can find the maximum spanning tree for us
  - Recursive algorithm
  - Naïve implementation: O(n^3)

Chu-Liu-Edmonds illustrated (for unlabeled dependency parsing)



Chu-Liu-Edmonds illustrated

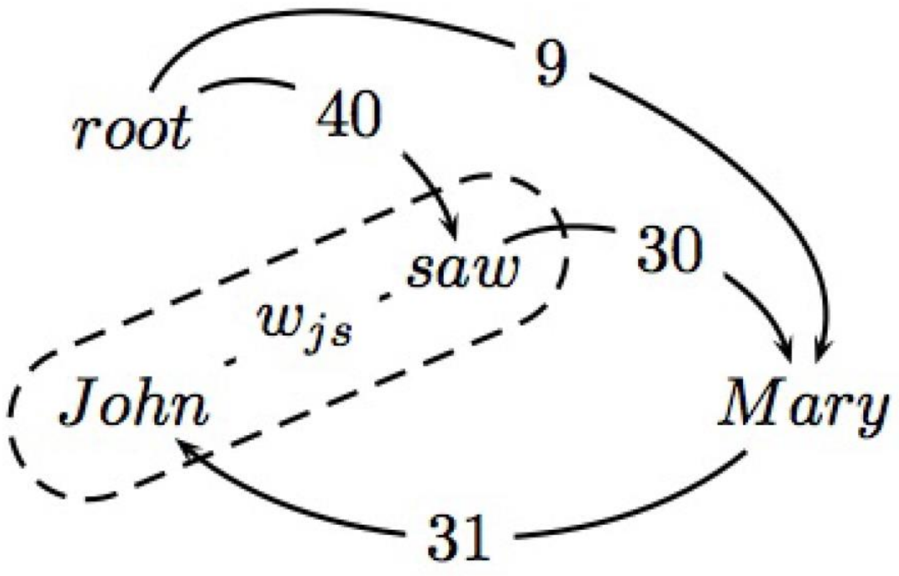
- Find highest scoring incoming arc for each vertex



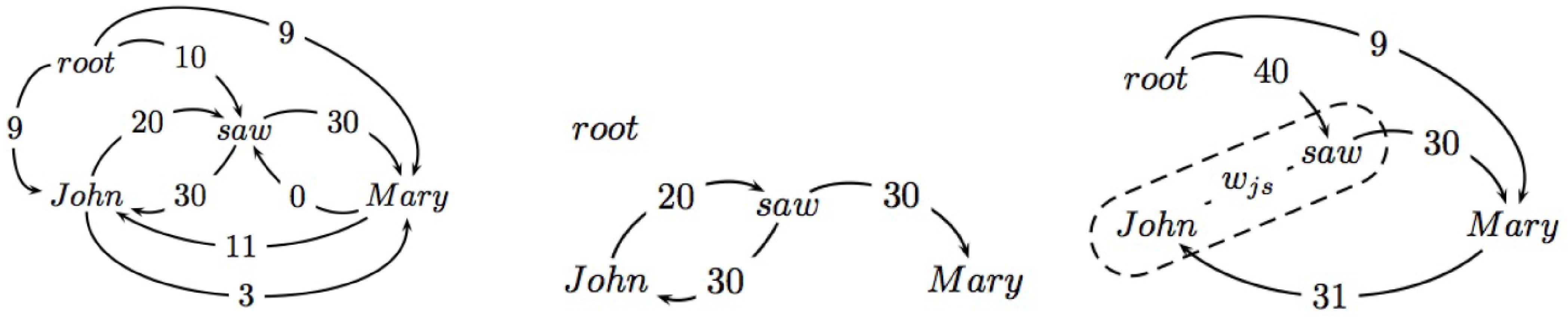
- If this is a tree, then we have found MST!!

Chu-Liu-Edmonds illustrated

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

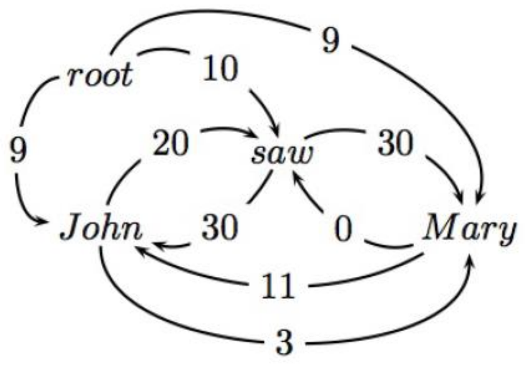


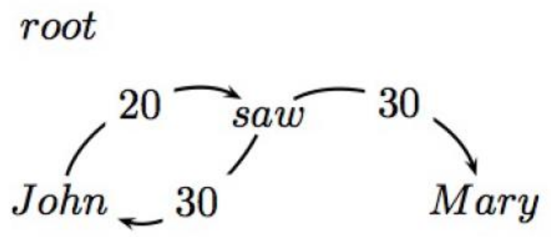
Chu-Liu-Edmonds illustrated

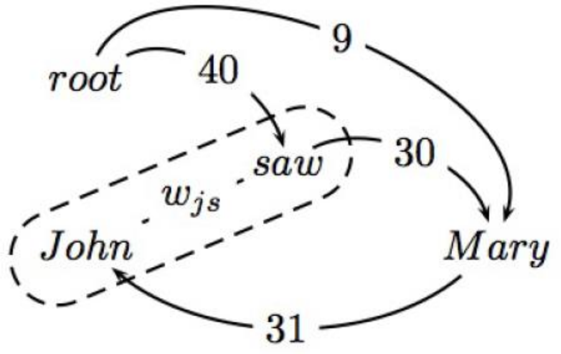


- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John $\rightarrow$ Mary is 3 and saw $\rightarrow$ Mary is 30

Chu-Liu-Edmonds illustrated

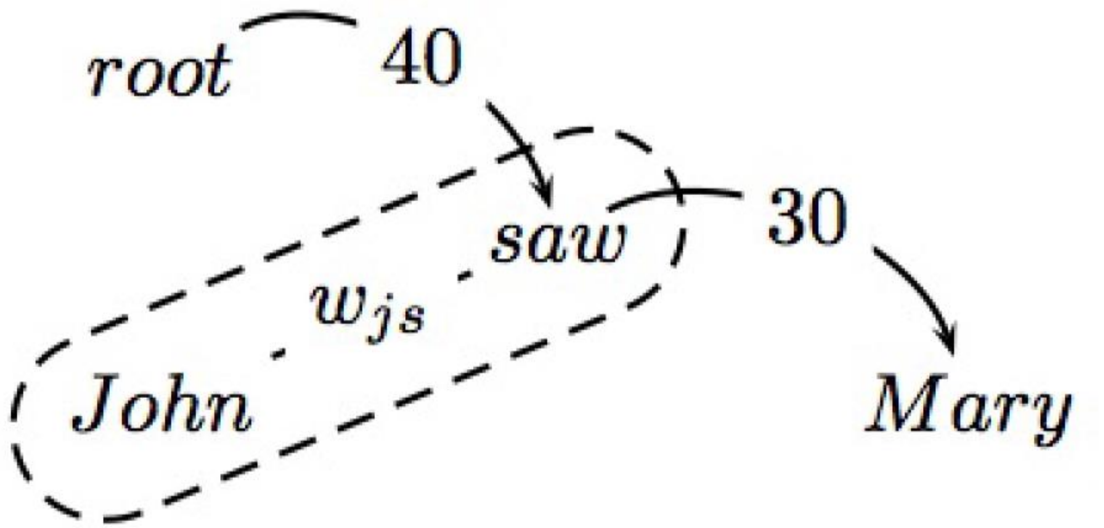






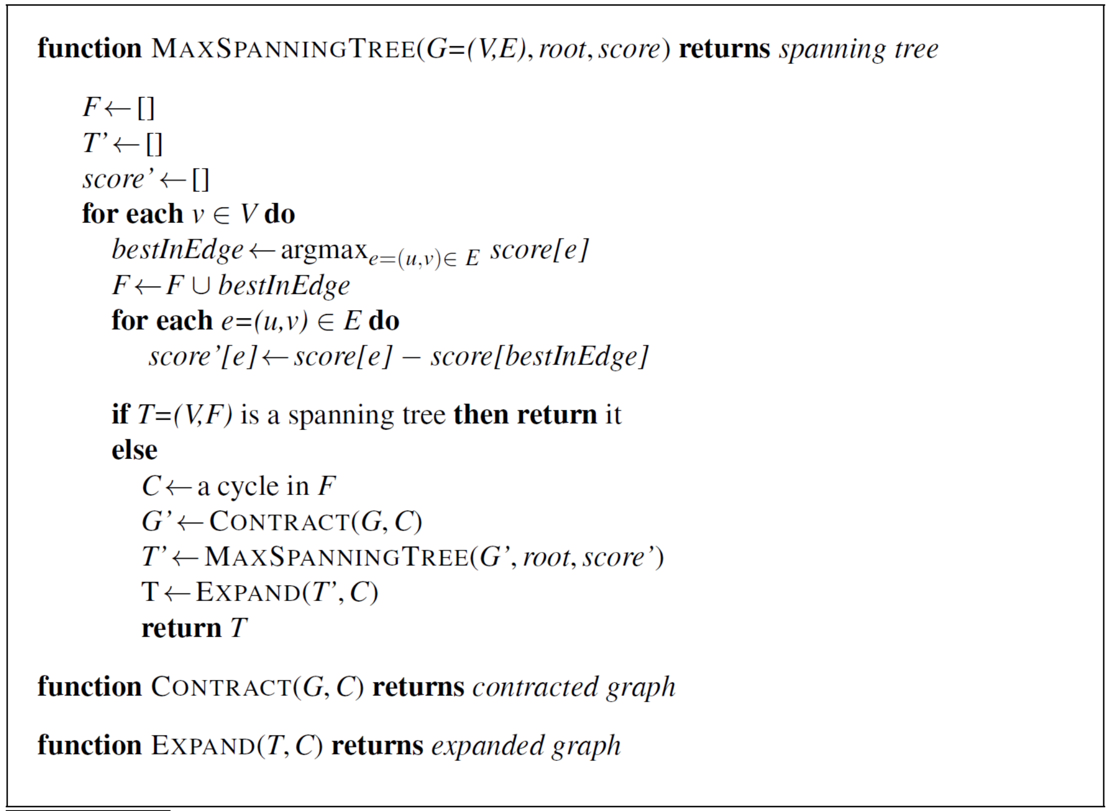
- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root $\longrightarrow$ saw $\longrightarrow$ John is 40 (**)
  - root $\longrightarrow$ John $\longrightarrow$ saw is 29

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



- Go back up recursive call and reconstruct final graph

Chu-Liu-Edmonds algorithm



```
function MAXSPANNINGTREE($G=(V,E)$, $root$, $score$) returns $spanning\ tree$

    $F \leftarrow []$
    $T' \leftarrow []$
    $score' \leftarrow []$
    for each $v \in V$ do
       $bestInEdge \leftarrow \text{argmax}_{e=(u,v) \in E}\ score[e]$
       $F \leftarrow F \cup bestInEdge$
       for each $e=(u,v) \in E$ do
           $score'[e] \leftarrow score[e] - score[bestInEdge]$

       if $T=(V,F)$ is a spanning tree then return it
       else
           $C \leftarrow$ a cycle in $F$
           $G' \leftarrow$ CONTRACT($G$, $C$)
           $T' \leftarrow$ MAXSPANNINGTREE($G'$, $root$, $score'$)
           $T \leftarrow$ EXPAND($T'$, $C$)
           return $T$

function CONTRACT($G$, $C$) returns $contracted\ graph$

function EXPAND($T$, $C$) returns $expanded\ graph$
```

Figure 15.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

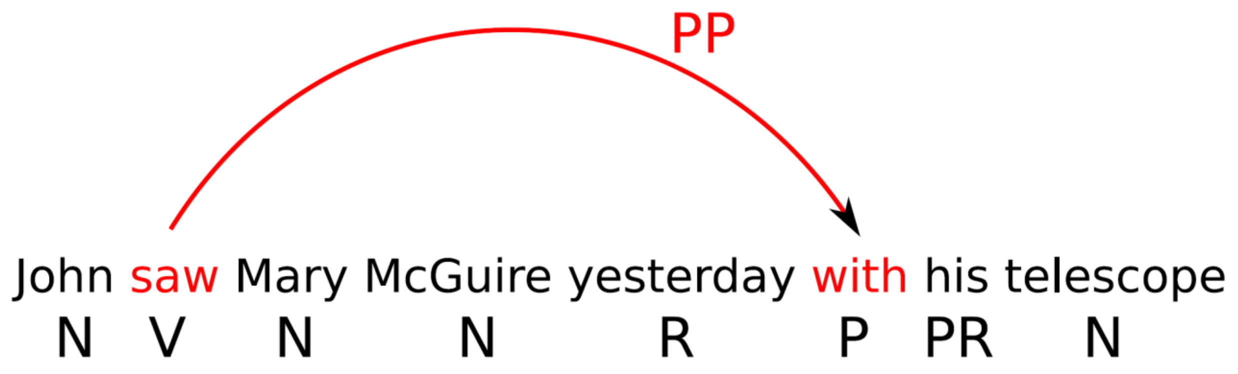
For dependency parsing, we will view arc weights as linear classifiers

Weight of arc from head **i** to dependent **j**, with label **k**

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

- Arc weights are a linear combination of features of the arc, **f**, and a corresponding weight vector **w**
- Raised to an exponent (simplifies some math ...)
- What arc 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

Typical classifier features

- Word forms, lemmas, and parts of speech of the headword and its dependent
- Corresponding features derived from the contexts before, after and between the words
- Word embeddings
- The dependency relation itself
- The direction of the relation (to the right or left)
- The distance from the head to the dependent
- …