Dependency Parsing & Feature-based Parsing

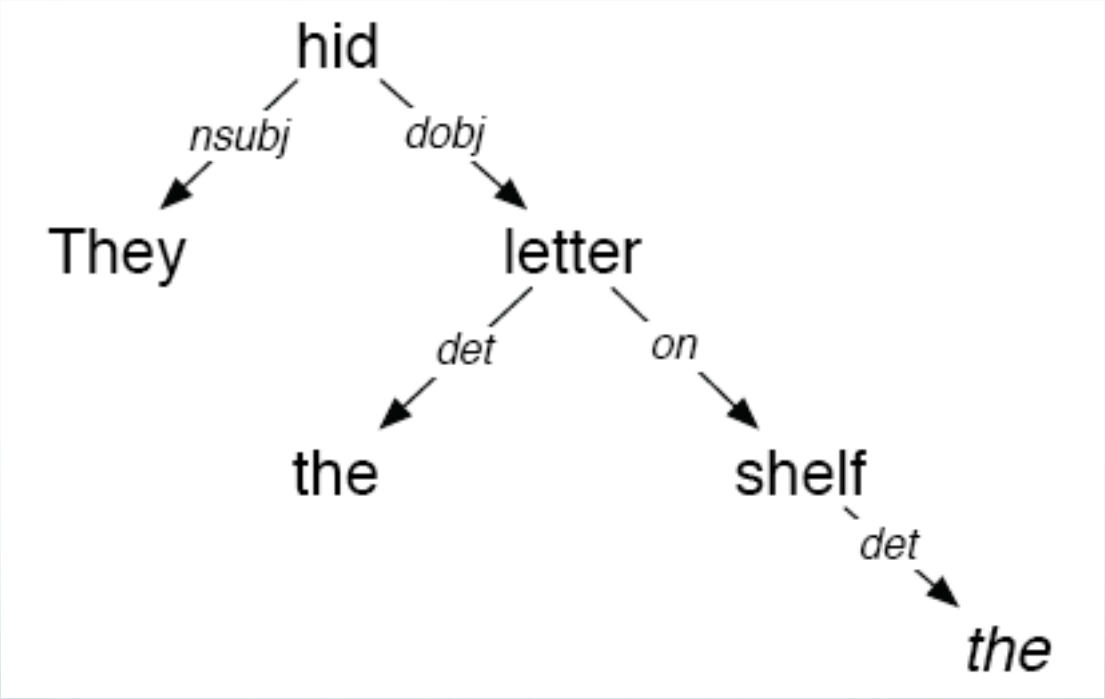
Ling571
Deep Processing Techniques for NLP
February 1, 2016

Roadmap

- Dependency parsing
  - Graph-based dependency parsing
    - Maximum spanning tree
    - CLE Algorithm
    - Learning weights
  - Transition-based parsing
    - Configurations and Oracles
- Feature-based parsing
  - Motivation
  - Features
  - Unification

Dependency Parse Example

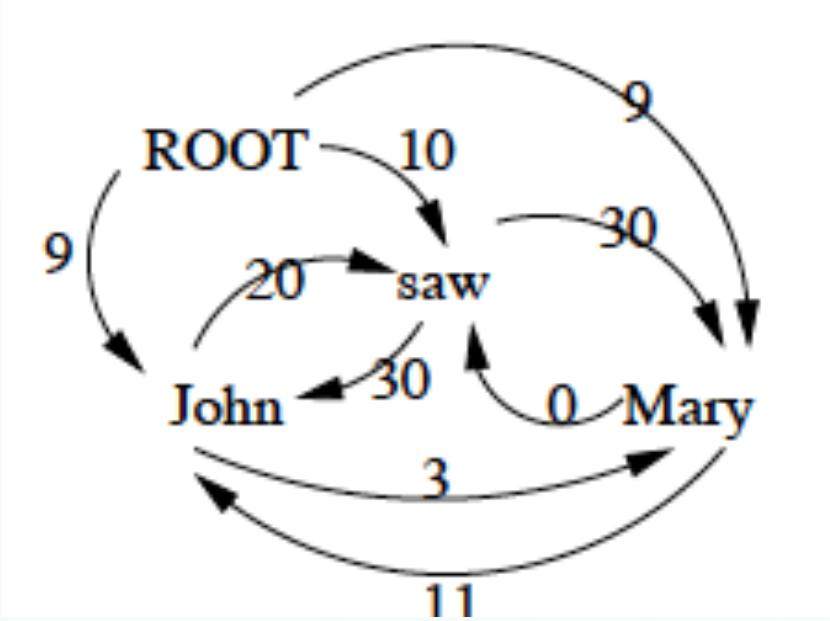
- They hid the letter on the shelf



Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree
- Idea:
  - Build initial graph: fully connected
    - Nodes: words in sentence to parse
    - Edges: Directed edges between all words
      - + Edges from ROOT to all words
  - Identify maximum spanning tree
    - Tree s.t. all nodes are connected
    - Select such tree with highest weight
    - Arc-factored model: Weights depend on end nodes & link
      - Weight of tree is sum of participating arcs

Initial Tree

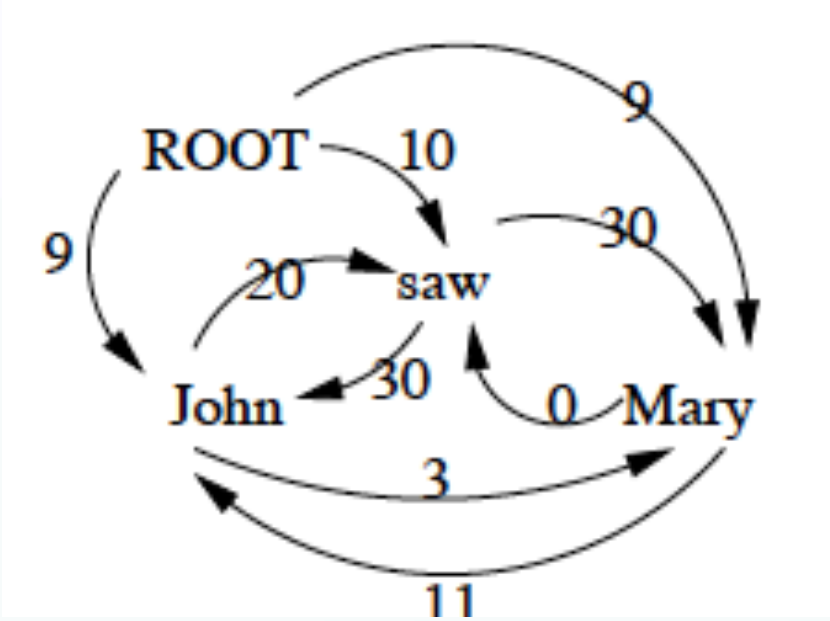


- Sentence: John saw Mary (McDonald et al, 2005)
  - All words connected; ROOT only has outgoing arcs
- Goal: Remove arcs to create a tree covering all words
  - Resulting tree is dependency parse

Maximum Spanning Tree

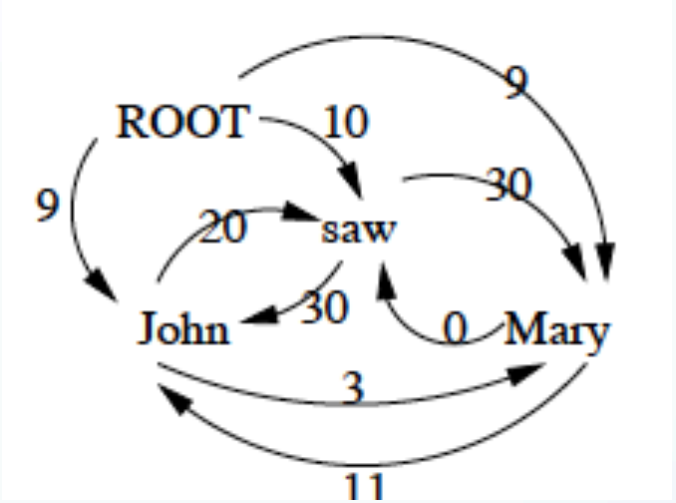
- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)
- Sketch of algorithm:
  - For each node, greedily select incoming arc with max w
  - If the resulting set of arcs forms a tree, this is the MST.
  - If not, there must be a cycle.
    - "Contract" the cycle: Treat it as a single vertex
    - Recalculate weights into/out of the new vertex
    - Recursively do MST algorithm on resulting graph
- Running time: naïve: $O(n^3)$; Tarjan: $O(n^2)$
  - Applicable to non-projective graphs

Initial Tree



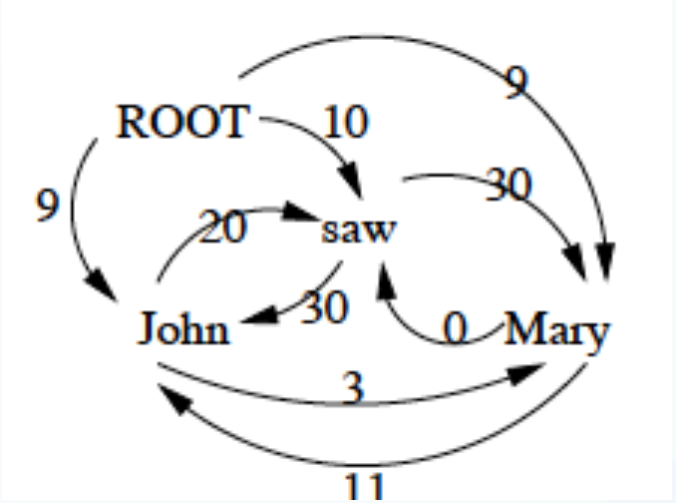
CLE: Step 1

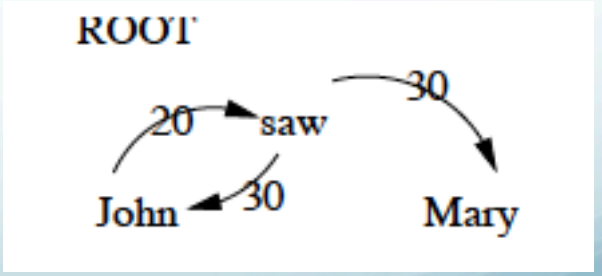
- Find maximum incoming arcs



CLE: Step 1

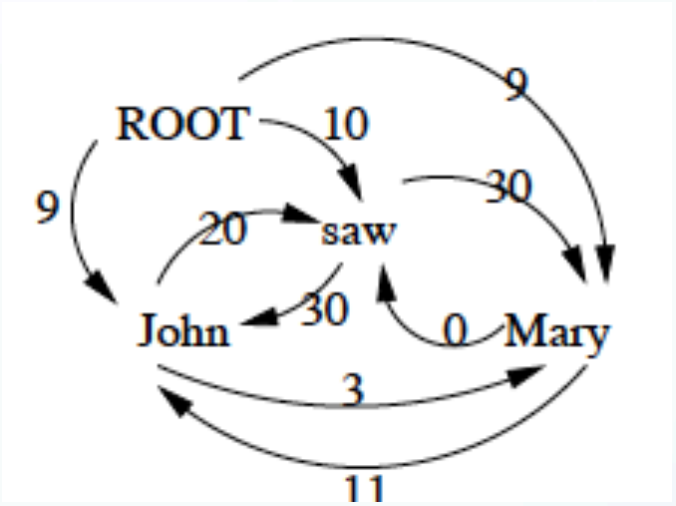
- Find maximum incoming arcs
  - Is the result a tree?

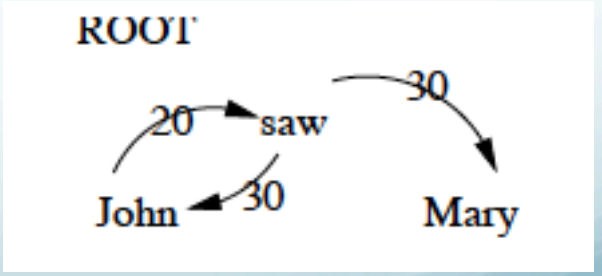




CLE: Step 1

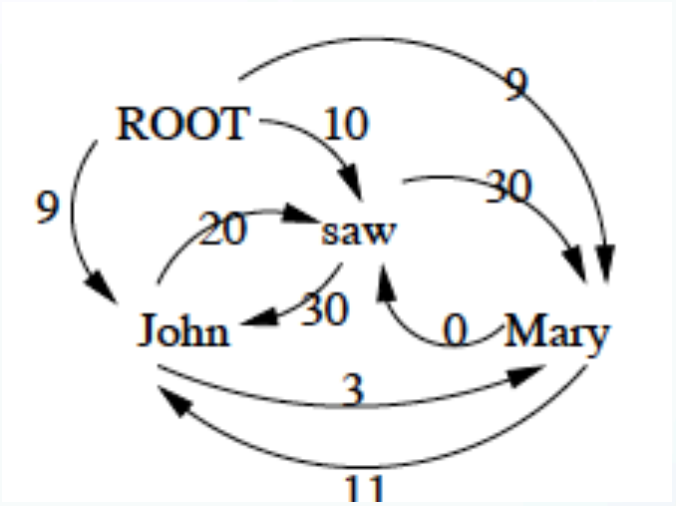
- Find maximum incoming arcs
  - Is the result a tree?
    - No
  - Is there a cycle?

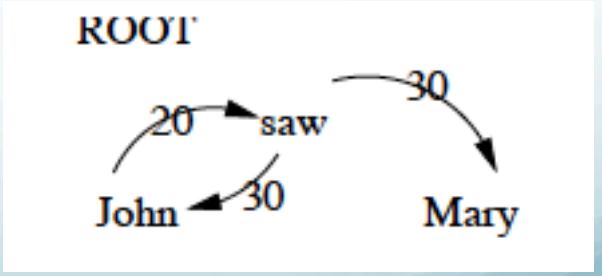




CLE: Step 1

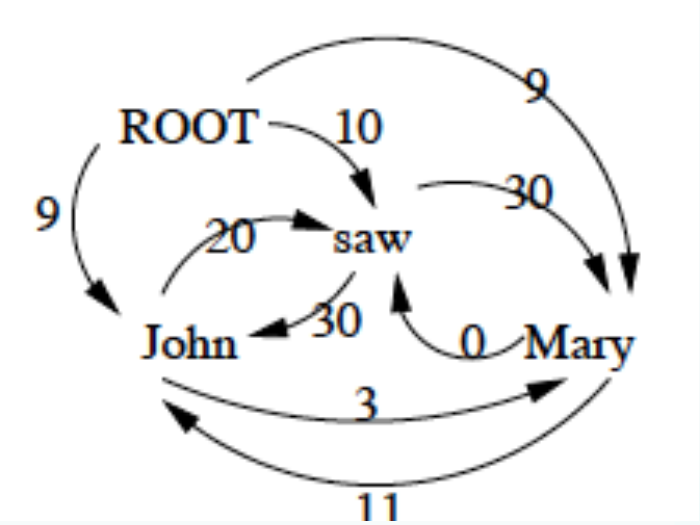
- Find maximum incoming arcs
  - Is the result a tree?
    - No
  - Is there a cycle?
    - Yes, John/saw





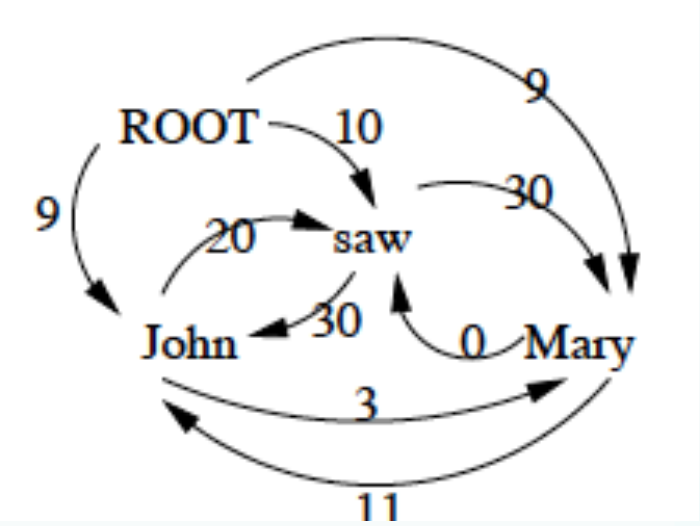
CLE: Step 2

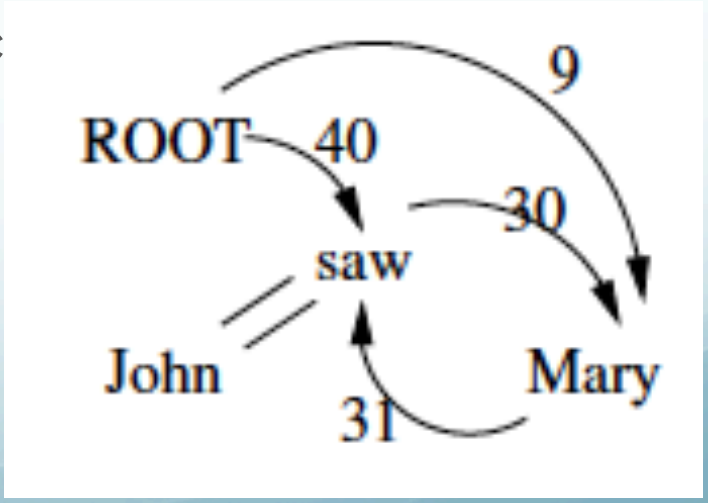
- Since there's a cycle:
  - Contract cycle & reweight
  - John+saw as single vertex



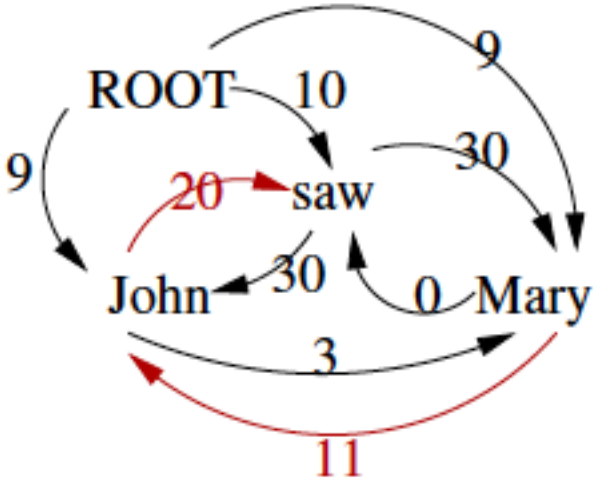
CLE: Step 2

- Since there's a cycle:
  - Contract cycle & reweight
  - John+saw as single vertex
  - Calculate weights in & out as:
    - Maximum based on internal arc
    - and original nodes
- Recurse

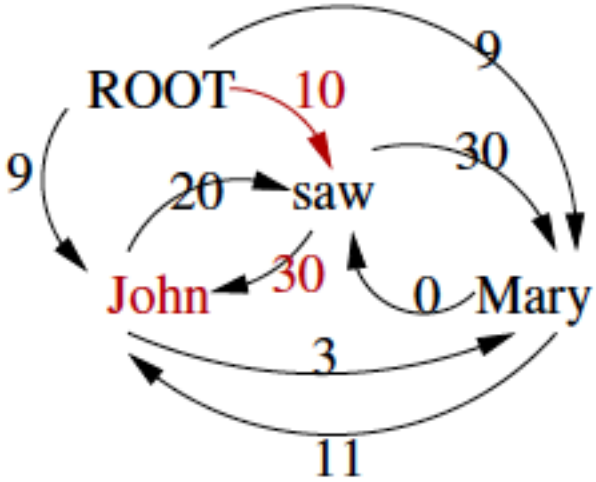




Calculating Graph



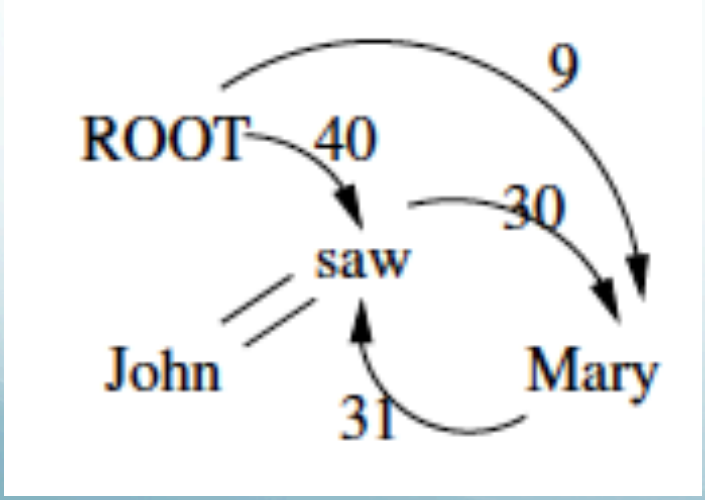
s(Mary, C) 11+20 = 31



s(ROOT, C) 10+30 = 40

CLE: Recursive Step

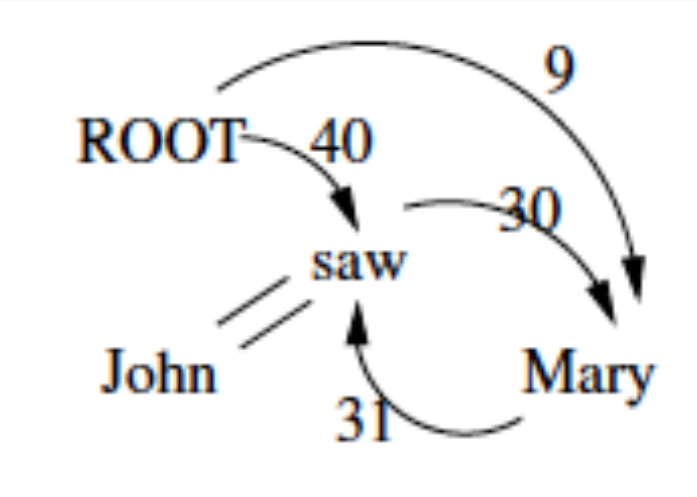
- In new graph, find graph of
  - Max weight incoming arc for each word

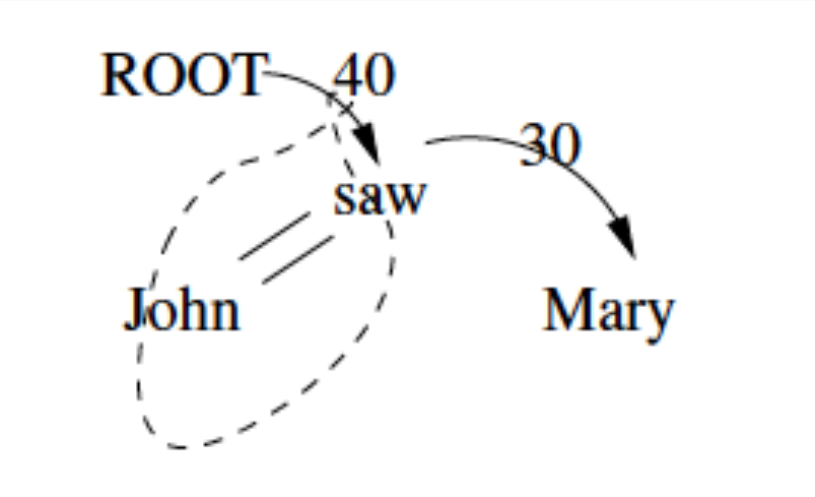


CLE: Recursive Step

- In new graph, find graph of
  - Max weight incoming arc for each word

- Is it a tree?

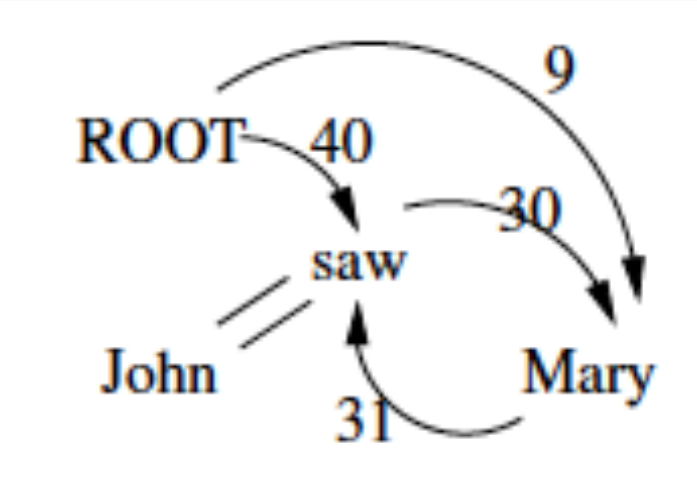


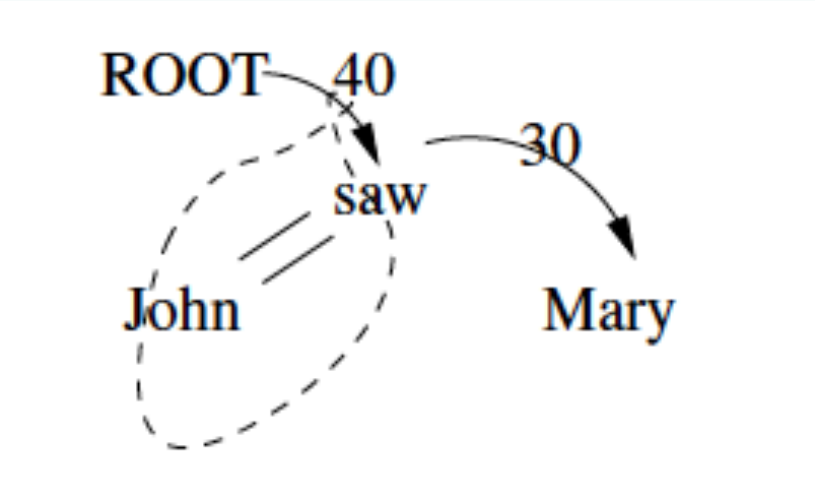


CLE: Recursive Step

- In new graph, find graph of
  - Max weight incoming arc for each word

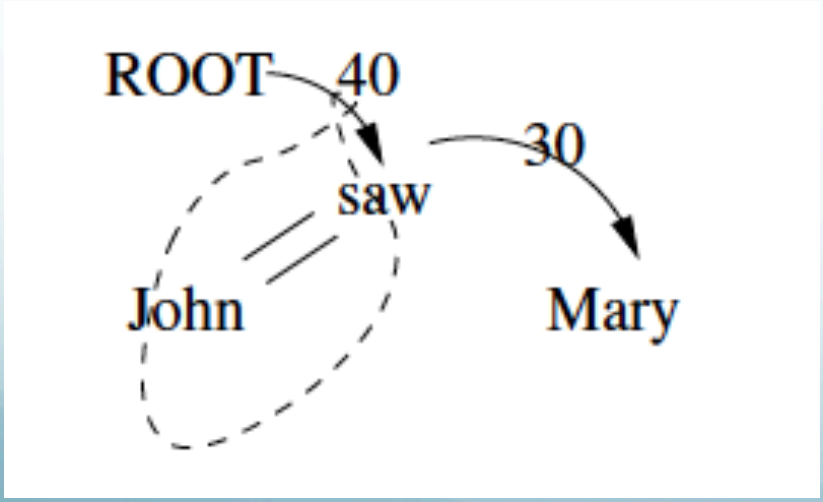
- Is it a tree? Yes!
  - MST, but must recover internal arcs ➔ parse





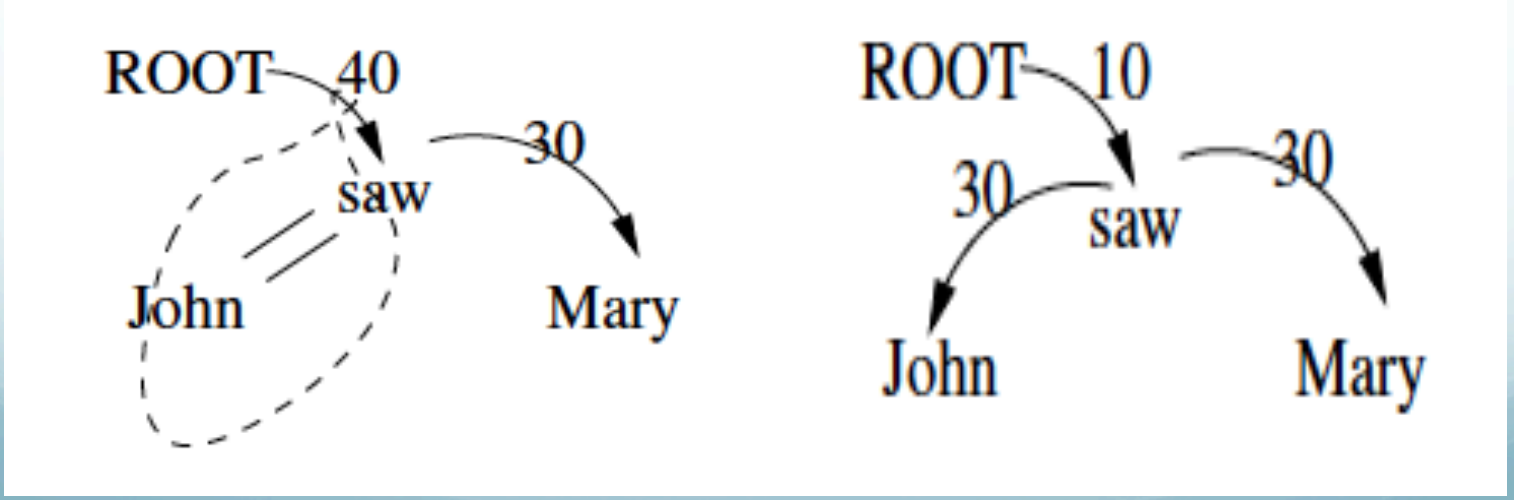
CLE: Recovering Graph

- Found maximum spanning tree
  - Need to ‘pop’ collapsed nodes
- Expand “ROOT → John+saw” = 40



CLE: Recovering Graph

- Found maximum spanning tree
  - Need to ‘pop’ collapsed nodes
- Expand “ROOT → John+saw” = 40
- MST and complete dependency parse



Learning Weights

- Weights for arc-factored model learned from corpus
  - Weights learned for tuple (w_i, L, w_j)

Learning Weights

- Weights for arc-factored model learned from corpus
  - Weights learned for tuple (w_i,L,w_j)
- McDonald et al, 2005 employed discriminative ML
  - Perceptron algorithm or large margin variant

Learning Weights

- Weights for arc-factored model learned from corpus
  - Weights learned for tuple (w_i,L,w_j)
- McDonald et al, 2005 employed discriminative ML
  - Perceptron algorithm or large margin variant
- Operates on vector of local features

Features for Learning Weights

- Simple categorical features for (w_i, L, w_j) including:
  - Identity of w_i (or char 5-gram prefix), POS of w_i
  - Identity of w_j (or char 5-gram prefix), POS of w_j
  - Label of L, direction of L
  - Sequence of POS tags b/t w_i, w_j
  - Number of words b/t w_i, w_j
  - POS tag of w_{i-1}, POS tag of w_{i+1}
  - POS tag of w_{j-1}, POS tag of w_{j+1}
- Features conjoined with direction of attachment and distance b/t words

Transition-based Parsing

- Parsing defined in terms of sequence of transitions
- Alternative methods for learning/decoding:
  - Most common model
    - Greedy classification-based approach

Transition-based Parsing

- Parsing defined in terms of sequence of transitions
- Alternative methods for learning/decoding:
  - Most common model
    - Greedy classification-based approach
- Very efficient method: O(n)
- Some of the most successful systems:
  - Highest scoring: 2008, (and current)

Transition-based Parsing

- Parsing defined in terms of sequence of transitions
- Alternative methods for learning/decoding:
  - Most common model
    - Greedy classification-based approach
- Very efficient method: O(n)
- Some of the most successful systems:
  - Highest scoring: 2008, (and current)
- Best-known implementations:
  - Nivre's MALTParser (2006, and onward)

Transition Systems

- A transition system for dependency parsing is:
  - A set of configurations
  - A set of transitions between configurations
  - An initialization function (for C_0)
  - A set of terminal configurations

Configurations

- A configuration for a sentence x is the triple (Σ ,B,A):
  - Σ is a stack with elements corresponding to the nodes (words + ROOT) in x
  - B (aka the buffer) is a list of nodes in x
  - A is the set of dependency arcs in the analysis so far,
    - (w_i,L,w_j), where w_x is a node in x, and L is a dep. label

Transitions

- Transitions convert one configuration to another s.t.
  - $C_i = t(C_{i-1})$, where t is the transition
- A dependency graph for a sentence is the set of arcs resulting from a sequence of transitions
- The parse of the sentence is that resulting from the initial state through the sequence of transitions to a legal terminal state.

Dependencies→Transitions

- To parse a sentence, we need the sequence of transitions that derives it.
- How can we determine sequence of transitions to parse a sentence?

Dependencies→Transitions

- To parse a sentence, we need the sequence of transitions that derives it.
- How can we determine sequence of transitions to parse a sentence?
  - Oracle: Given a dependency parse, identifies transition sequence.
    - Nivre's Arc-standard parser: provably sound & complete

Dependencies→Transitions

- To parse a sentence, we need the sequence of transitions that derives it.
- How can we determine sequence of transitions to parse a sentence?
  - Oracle: Given a dependency parse, identifies transition sequence.
    - Nivre's Arc-standard parser: provably sound & complete
  - Training:
    - Use oracle method on dependency treebank
      - Identifies gold transitions for configurations
    - Train classifier to predict best transition in new config.

Nivre's Arc-Standard Oracle

- Words: w_1,....,w_n; Index 0: "dummy root"
- Initialization:
  - Stack = [0]; Buffer = [1,..,n]; Arcs = Ø

Nivre's Arc-Standard Oracle

- Words: w_1,....,w_n; Index 0: "dummy root"
- Initialization:
  - Stack = [0]; Buffer = [1,..,n]; Arcs = Ø
- Termination:
  - Stack = [0]; Buffer = []; Arcs = A

Nivre's Arc-Standard Oracle

- Words: w_1,....,w_n; Index 0: "dummy root"
- Initialization:
  - Stack = [0]; Buffer = [1,..,n]; Arcs = Ø
- Termination:
  - Stack = [0]; Buffer = []; Arcs = A
- Transitions:
  - Shift: Push first element of buffer on top of stack
    - [i][j,k,,n][] ➔ [i|j][k,...,n][]

Nivre's Arc-Standard Oracle

- Transitions: Left-Arc label
  - Make left-arc between top two elements of stack
    - Remove dependent from stack, add arc to A
  - [i|j] [k,…,n] A → [j] [k,…n] A U (j, label, i)
    - i not root

Nivre's Arc-Standard Oracle

- Transitions: Left-Arc label
  - Make left-arc between top two elements of stack
    - Remove dependent from stack, add arc to A
  - [i|j] [k,...,n] A → [j] [k,...n] A U (j, label, i)
    - I not root
- Transitions: Right-Arc label
  - Make right-arc between top two elements of stack
    - Remove dependent from stack, add arc to A
  - [i|j] [k,...,n] A → [i] [k,...n] A U (i, label, j)

Nivre's Arc-Standard Oracle

- Algorithm:
- Initialize configuration
- While (Buffer not empty) and (stack not only root):
  - If (top of stack is head of 2nd in stack) and (all children of 2nd attached), then Left-Arc
  - If (2nd in stack is head of top of stack) and (all children of top are attached), then Right-Arc
  - Else shift

Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]

Example (Ballesteros '15)

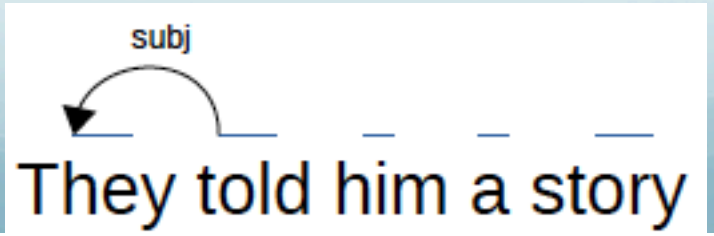
Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]

Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]

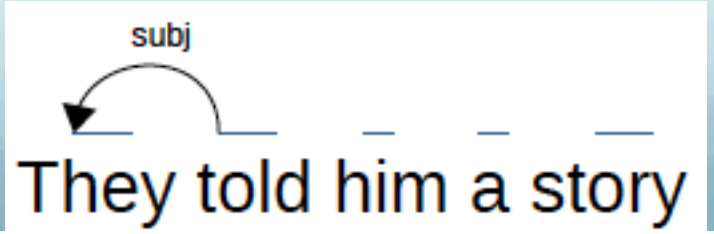
Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]
Left-Arc (subj)	[told]	[him a story)



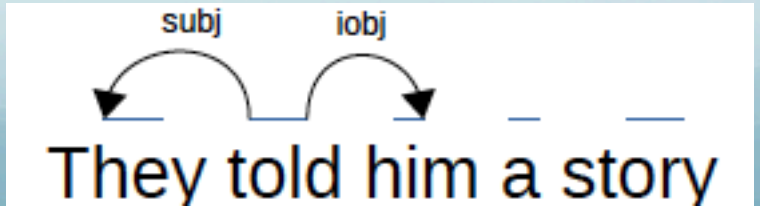
Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]
Left-Arc (subj)	[told]	[him a story)
Shift	[told, him]	[a story]



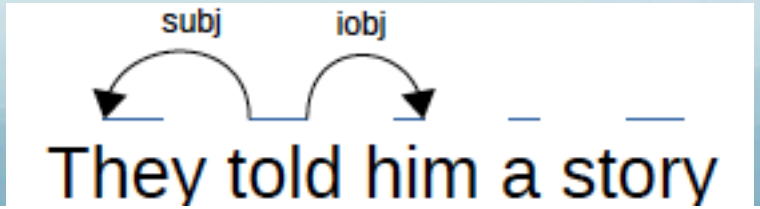
Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]
Left-Arc (subj)	[told]	[him a story)
Shift	[told, him]	[a story]
Right-Arc (iobj)	[told]	[a story]



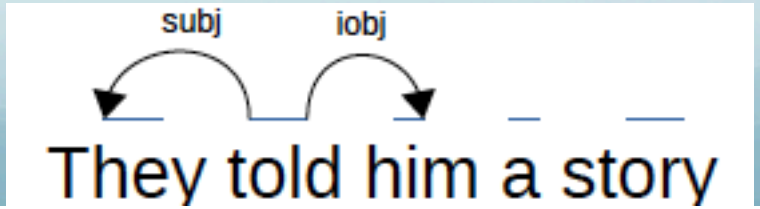
Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]
Left-Arc (subj)	[told]	[him a story)
Shift	[told, him]	[a story]
Right-Arc (iobj)	[told]	[a story]
Shift	[told, a]	[story]



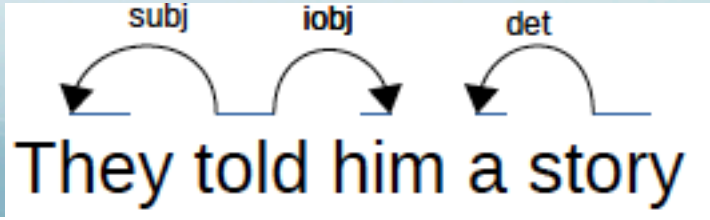
Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]
Left-Arc (subj)	[told]	[him a story)
Shift	[told, him]	[a story]
Right-Arc (iobj)	[told]	[a story]
Shift	[told, a]	[story]
Shift	[told, a, story]	[]



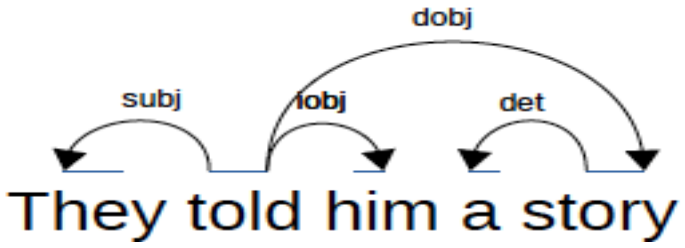
Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]
Left-Arc (subj)	[told]	[him a story)
Shift	[told, him]	[a story]
Right-Arc (iobj)	[told]	[a story]
Shift	[told, a]	[story]
Shift	[told, a, story]	[]
Left-Arc (Det)	[told, story]	[]



Example (Ballesteros '15)

Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]
Shift	[They, told]	[him a story]
Left-Arc (subj)	[told]	[him a story)
Shift	[told, him]	[a story]
Right-Arc (iobj)	[told]	[a story]
Shift	[told, a]	[story]
Shift	[told, a, story]	[]
Left-Arc (Det)	[told, story]	[]
Right-Arc (dobj)	[told]	[]



Transition Classification

- Train classifier to predict transitions:
  - {Left-Arc,Right-Arc} x Number of dependency labels

Transition Classification

- Train classifier to predict transitions:
  - {Left-Arc,Right-Arc) x Number of dependency labels
- Features:
  - Different components of configuration
    - Word, POS of stack, buffer positions
    - Previous labels/transitions

Transition Classification

- Train classifier to predict transitions:
  - {Left-Arc,Right-Arc) x Number of dependency labels
- Features:
  - Different components of configuration
    - Word, POS of stack, buffer positions
    - Previous labels/transitions
- Classifier: Any
  - Original work: SVMs; Currently: DNNs + LSTM

Dependency Parsing

- Dependency grammars:
  - Compactly represent pred-arg structure
  - Lexicalized, localized
  - Natural handling of flexible word order

Dependency Parsing

- Dependency grammars:
  - Compactly represent pred-arg structure
  - Lexicalized, localized
  - Natural handling of flexible word order
- Dependency parsing:
  - Conversion to phrase structure trees
  - Graph-based parsing (MST), efficient non-proj $O(n^2)$
  - Transition-based parser
    - MALTparser: very efficient O(n)
      - Optimizes local decisions based on many rich features

Features

Roadmap

- Features: Motivation
  - Constraint & compactness
- Features
  - Definitions & representations
- Unification
- Application of features in the grammar
  - Agreement, subcategorization
- Parsing with features & unification
  - Augmenting the Earley parser, unification parsing
- Extensions: Types, inheritance, etc
- Conclusion

Constraints & Compactness

- Constraints in grammar
  - S → NP VP
    - They run.
    - He runs.

Constraints & Compactness

- Constraints in grammar
  - S → NP VP
    - They run.
    - He runs.
  - But…
    - *They runs
    - *He run
    - *He disappeared the flight

Constraints & Compactness

- Constraints in grammar
  - S → NP VP
    - They run.
    - He runs.
  - But…
    - *They runs
    - *He run
    - *He disappeared the flight
  - Violate agreement (number), subcategorization

Enforcing Constraints

- Enforcing constraints

Enforcing Constraints

- Enforcing constraints
  - Add categories, rules

Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - S→ NPsg3p VPsg3p,
      - S→ NPpl3p VPpl3p,

Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - S→ NPsg3p VPsg3p,
      - S→ NPpl3p VPpl3p,
    - Subcategorization:
      - VP→ Vtrans NP,
      - VP → Vintrans,
      - VP → Vditrans NP NP

Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - S→ NPsg3p VPsg3p,
      - S → NPpl3p VPpl3p,
    - Subcategorization:
      - VP → Vtrans NP,
      - VP → Vintrans,
      - VP → Vditrans NP NP
  - Explosive!, loses key generalizations

Why features?

- Need compact, general constraints
  - S → NP VP

Why features?

- Need compact, general constraints
  - S → NP VP
    - Only if NP and VP agree

Why features?

- Need compact, general constraints
  - S → NP VP
    - Only if NP and VP agree
- How can we describe agreement, subcat?

Why features?

- Need compact, general constraints
  - S → NP VP
    - Only if NP and VP agree
- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement

Why features?

- Need compact, general constraints
  - S → NP VP
    - Only if NP and VP agree
- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
    - Number, person, gender, etc

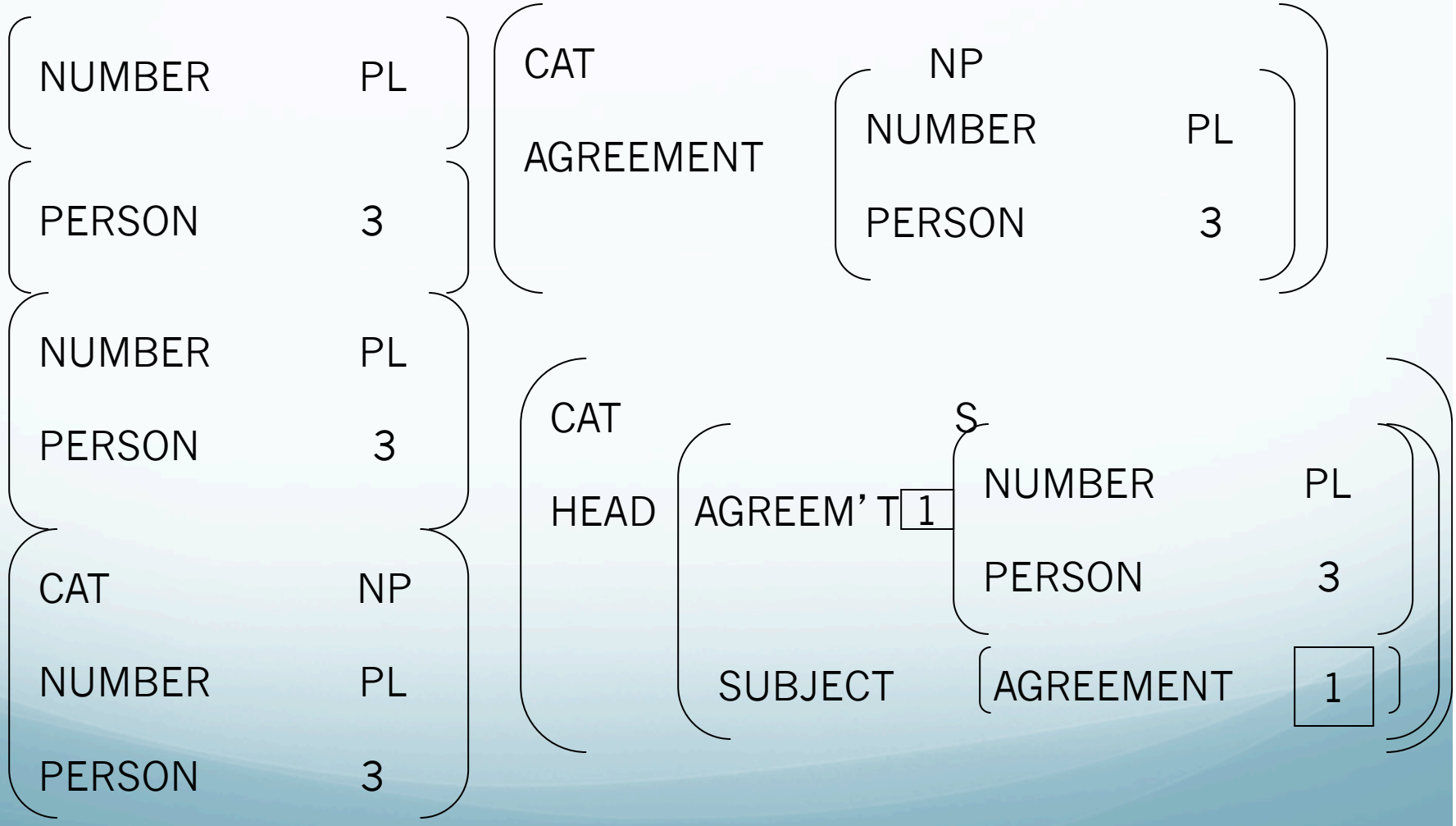
Why features?

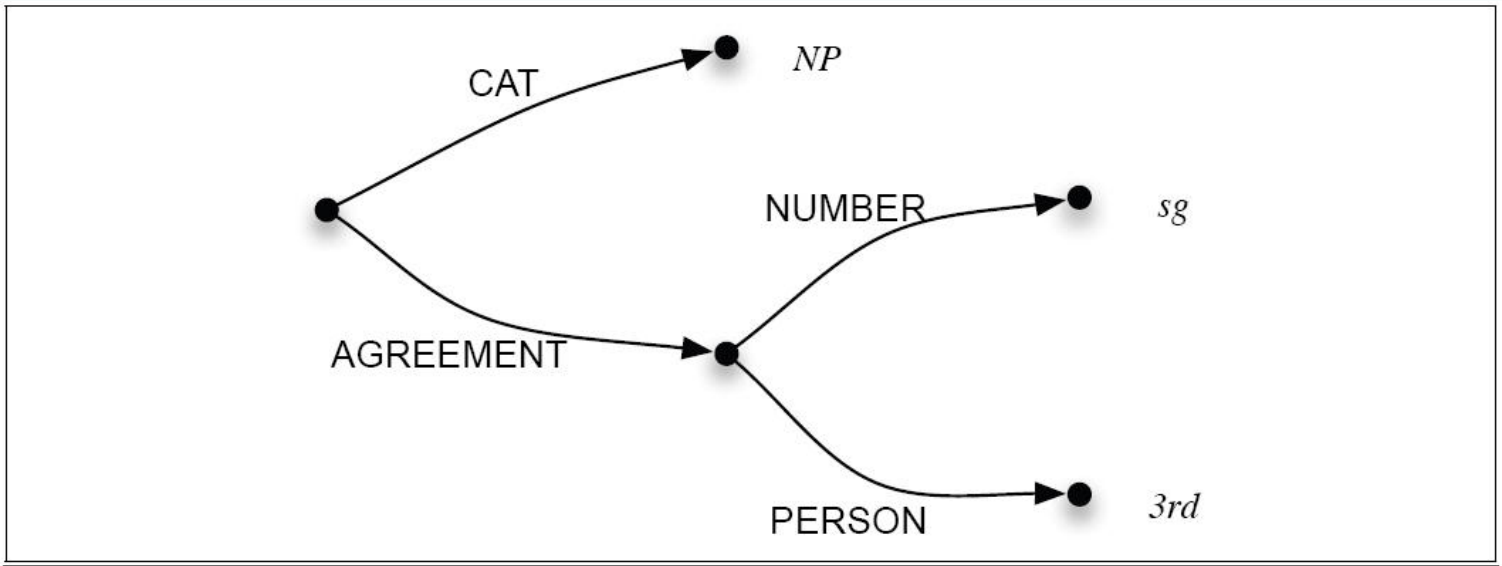
- Need compact, general constraints
  - S → NP VP
    - Only if NP and VP agree
- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
    - Number, person, gender, etc
- Augment CF rules with feature constraints
  - Develop mechanism to enforce consistency
  - Elegant, compact, rich representation

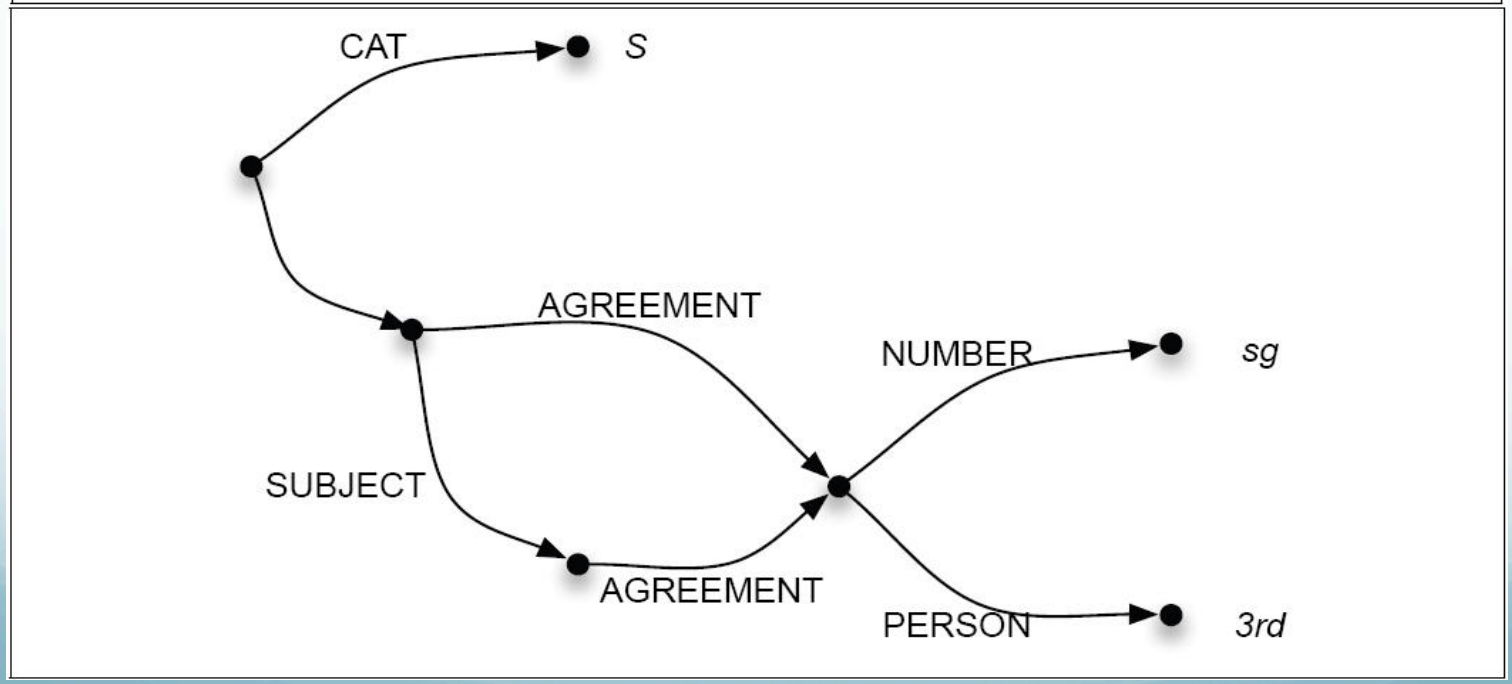
Feature Representations

- Fundamentally, Attribute-Value pairs
  - Values may be symbols or feature structures
    - Feature path: list of features in structure to value
    - “Reentrant feature structures”: share some struct
  - Represented as
    - Attribute-value matrix (AVM), or
    - Directed acyclic graph (DAG)

AVM


NUMBER PL
PERSON 3
NUMBER PL
PERSON 3
CAT NP
NUMBER PL
PERSON 3
CAT NP
AGREEMENT
NUMBER PL
PERSON 3
CAT S
HEAD
AGREEM' T 1
NUMBER PL
PERSON 3
SUBJECT
AGREEMENT 1


CAT
NP
NUMBER
sg
AGREEMENT
PERSON
3rd


CAT
S
AGREEMENT
NUMBER
sg
SUBJECT
AGREEMENT
PERSON
3rd

Unification

- Two key roles:

Unification

- Two key roles:
  - Merge compatible feature structures

Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if

Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
  - Feature structures are identical
    - Result in same structure

Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
  - Feature structures are identical
    - Result in same structure
  - Feature structures match where both have values, differ in missing or underspecified
    - Resulting structure incorporates constraints of both

Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)

Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)
- Examples:
  - A: [Number SG], B: [Person 3]
  - C:[Number SG]
    - [Person 3]

Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)
- Examples:
  - A: [Number SG], B: [Person 3]
  - C:[Number SG]
    - [Person 3]
  - A subsumes C; B subsumes C; B,A don't subsume
    - Partial order on f.s.

Unification Examples

- Identical
  - [Number SG] U [Number SG]

Unification Examples

- Identical
  - [Number SG] U [Number SG]=[Number SG]
- Underspecified
  - [Number SG] U [Number []]

Unification Examples

- Identical
  - [Number SG] U [Number SG]=[Number SG]
- Underspecified
  - [Number SG] U [Number []] = [Number SG]
- Different specification
  - [Number SG] U [Person 3]

Unification Examples

- Identical
  - [Number SG] U [Number SG]=[Number SG]
- Underspecified
  - [Number SG] U [Number []] = [Number SG]
- Different specification
  - [Number SG] U [Person 3] = [Number SG]
  - [Person 3]
  - [Number SG] U [Number PL]

Unification Examples

- Identical
  - [Number SG] U [Number SG]=[Number SG]
- Underspecified
  - [Number SG] U [Number []] = [Number SG]
- Different specification
  - [Number SG] U [Person 3] = [Number SG]
  - [Person 3]
- Mismatched
  - [Number SG] U [Number PL] → Fails!

More Unification Examples

$$\begin{bmatrix} \text{AGREEMENT} & [1] \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & [1] \end{bmatrix} \end{bmatrix} \quad \cup$$

$$\begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad =$$

$$\begin{bmatrix} \text{AGREEMENT} & [1] \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & [1] & \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$