Dependency & Feature-Based Parsing

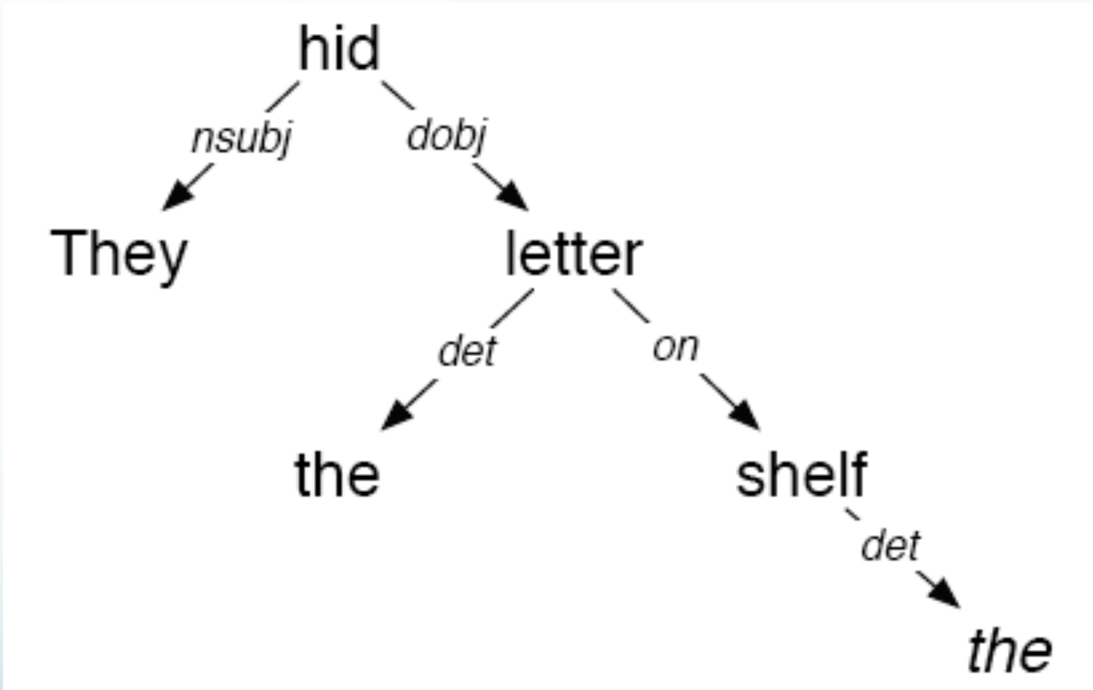
Deep Processing for NLP
Ling571
February 3, 2014

Roadmap

- Dependency Parsing:
  - Convert dependency trees to PS trees
    - Parse using standard algorithms $O(n^3)$
  - Employ graph-based optimization
    - Weights learned by machine learning
  - Shift-reduce approaches based on current word/state
    - Attachment based on machine learning

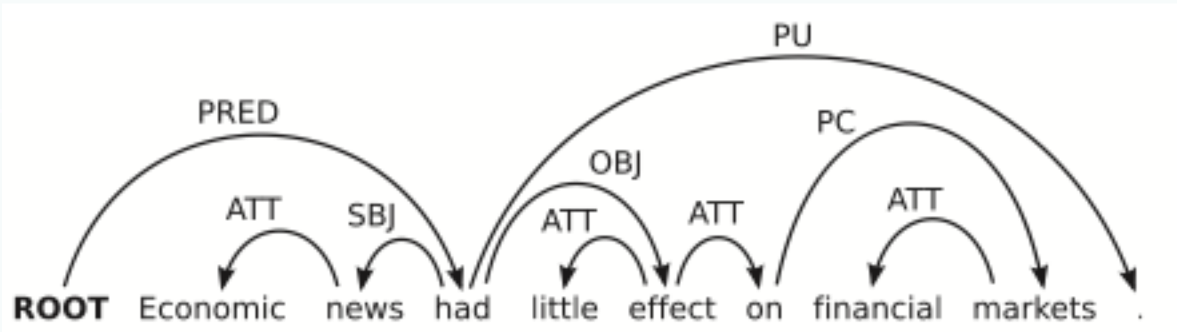
Dependency Parse Example

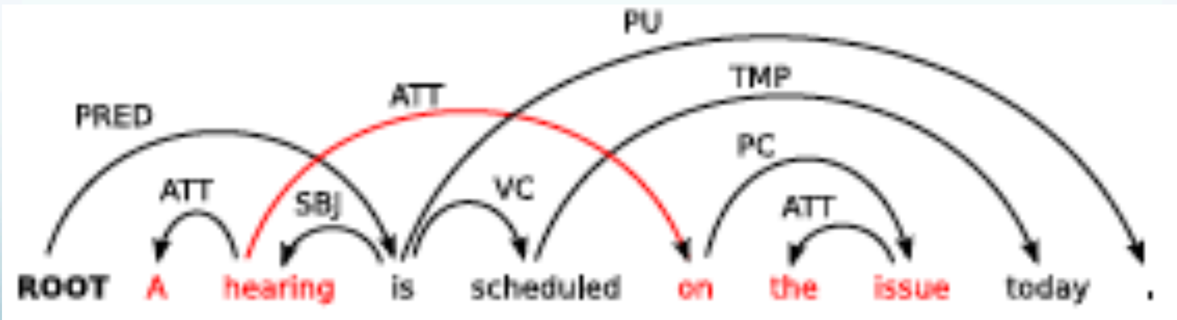
- They hid the letter on the shelf



Parsing by PS Conversion

- Can map any projective dependency tree to PS tree
  - Non-terminals indexed by words
    - "Projective": no crossing dependency arcs for ordered words



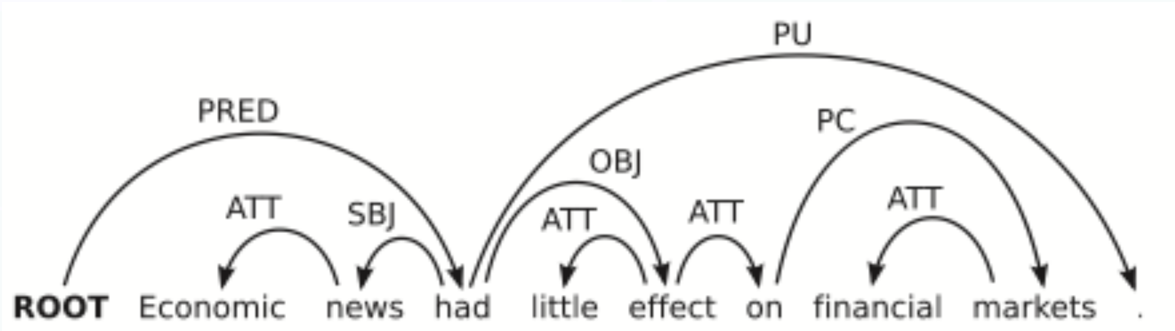


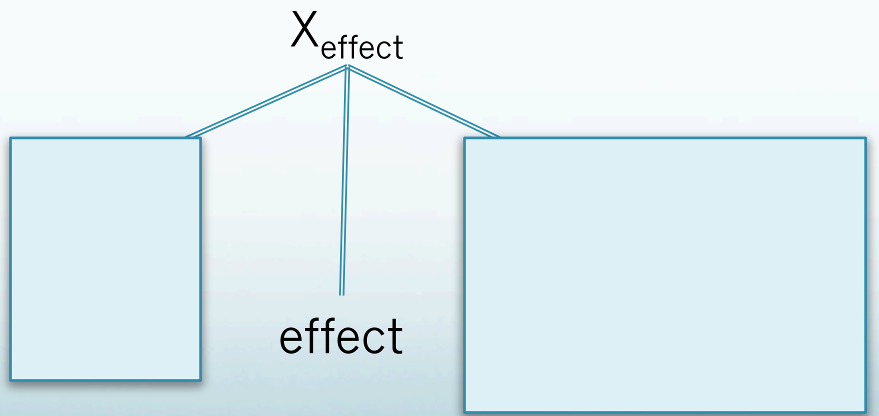
Dep to PS Tree Conversion

- For each node *w* with outgoing arcs,
  - Convert the subtree *w* and its dependents $t_1,..,t_n$ to
  - New subtree rooted at X_w with child *w* and
    - Subtrees at $t_1,..,t_n$ in the original sentence order

Dep to PS Tree Conversion

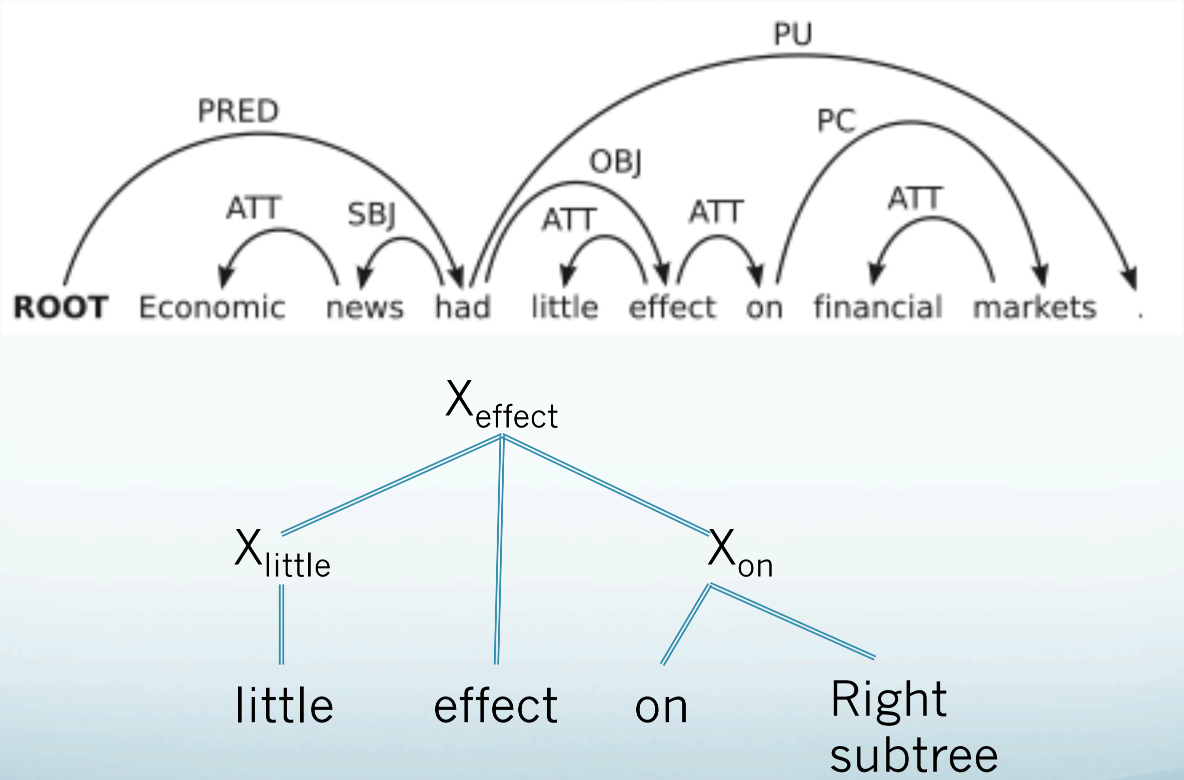
E.g., for ‘effect’





Dep to PS Tree Conversion

E.g., for 'effect'



PS to Dep Tree Conversion

- What about the dependency labels?
  - Attach labels to non-terminals associated with non-heads
  - E.g. $X_{little} \rightarrow X_{little:nmod}$

PS to Dep Tree Conversion

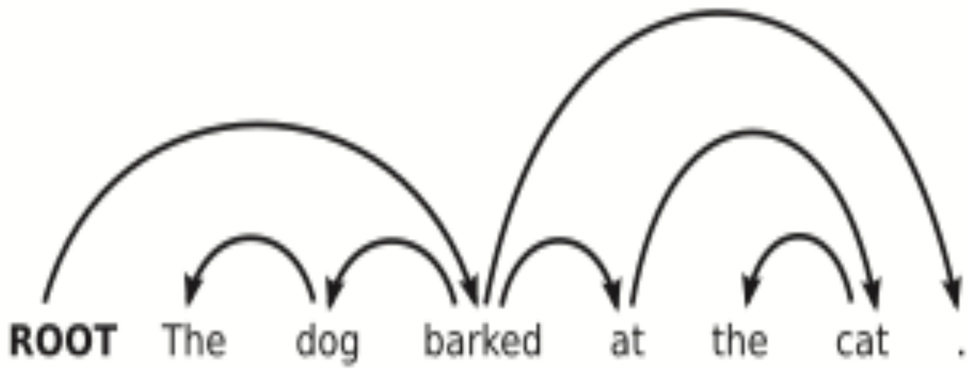
- What about the dependency labels?
  - Attach labels to non-terminals associated with non-heads
  - E.g. $X_{little} \rightarrow X_{little:nmod}$

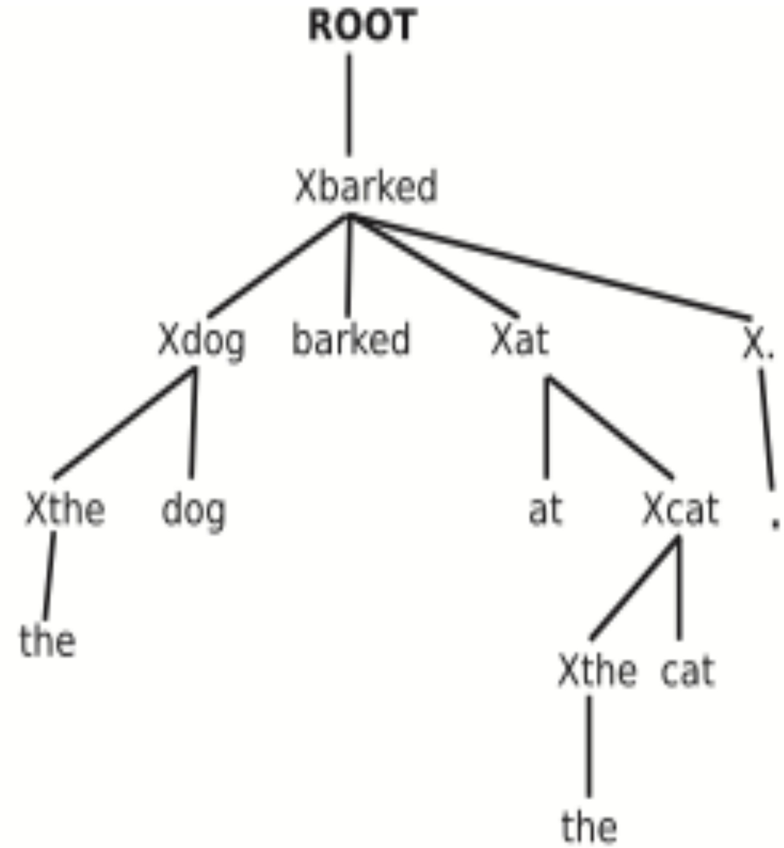
- Doesn't create typical PS trees
  - Does create fully lexicalized, context-free trees
    - Also labeled

PS to Dep Tree Conversion

- What about the dependency labels?
  - Attach labels to non-terminals associated with non-heads
  - E.g. $X_{little} \rightarrow X_{little:nmod}$
- Doesn't create typical PS trees
  - Does create fully lexicalized, context-free trees
    - Also labeled
- Can be parsed with any standard CFG parser
  - E.g. CKY, Earley

Full Example Trees





Example from J. Moore, 2013

Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree T for sentence S
  - If S is unambiguous, T is the correct parse.
  - If S is ambiguous, T is the highest scoring parse.

Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree T for sentence S
  - If S is unambiguous, T is the correct parse.
  - If S is ambiguous, T is the highest scoring parse.
- Where do scores come from?
  - Weights on dependency edges by machine learning
  - Learned from large dependency treebank

Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree T for sentence S
  - If S is unambiguous, T is the correct parse.
  - If S is ambiguous, T is the highest scoring parse.
- Where do scores come from?
  - Weights on dependency edges by machine learning
  - Learned from large dependency treebank
- Where are the grammar rules?

Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree T for sentence S
  - If S is unambiguous, T is the correct parse.
  - If S is ambiguous, T is the highest scoring parse.
- Where do scores come from?
  - Weights on dependency edges by machine learning
  - Learned from large dependency treebank
- Where are the grammar rules?
  - There aren't any; data-driven processing

Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree

Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree
- Idea:
  - Build initial graph: fully connected
    - Nodes: words in sentence to parse

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

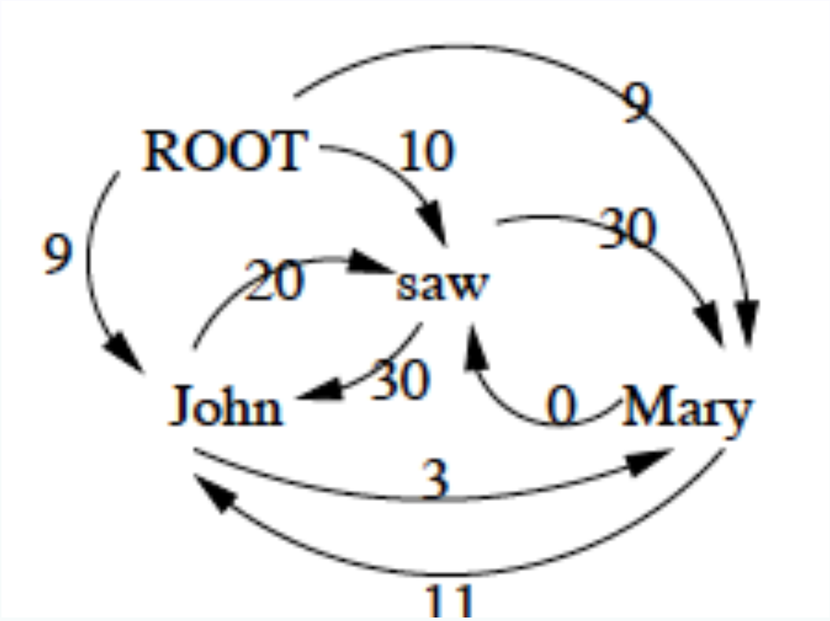
Graph-based Dependency Parsing

- 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

Graph-based Dependency Parsing

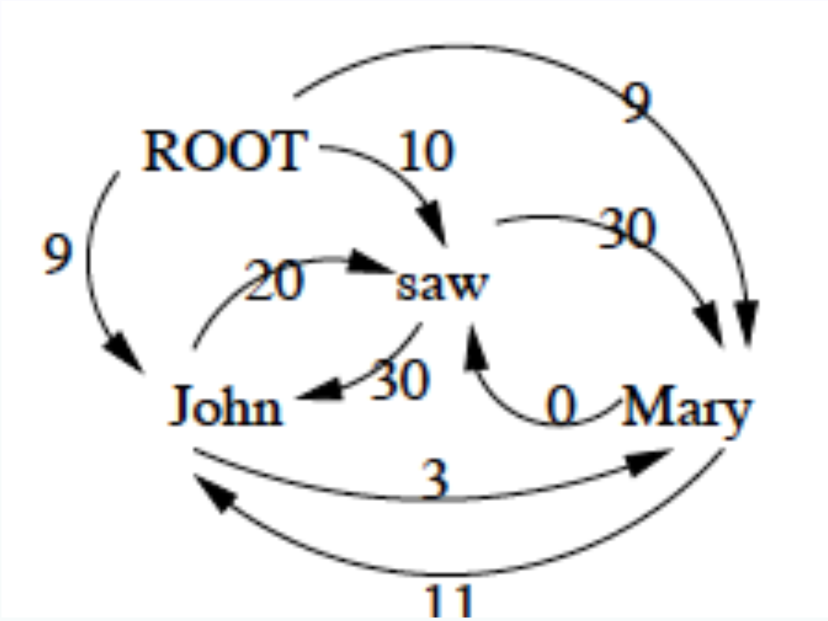
- Map dependency parsing to maximum spanning tree
- Idea:
  - Build initial graph: fully connected
    - Nodes: words in sentence to parse
    - Edges: Directed edges between all words
      - + Edges from ROOT to all words
  - Identify maximum spanning tree
    - Tree s.t. all nodes are connected
    - 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

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)

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.

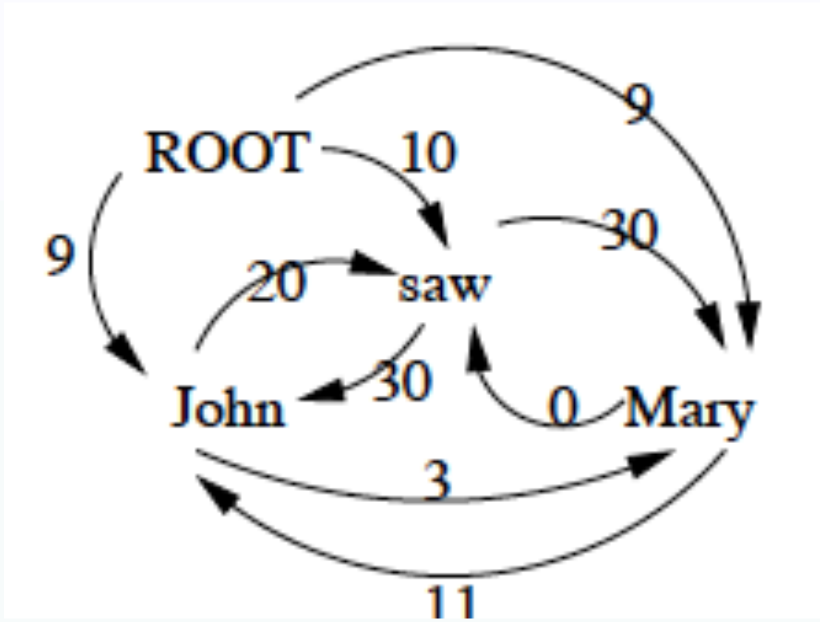
Maximum Spanning Tree

- 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

Maximum Spanning Tree

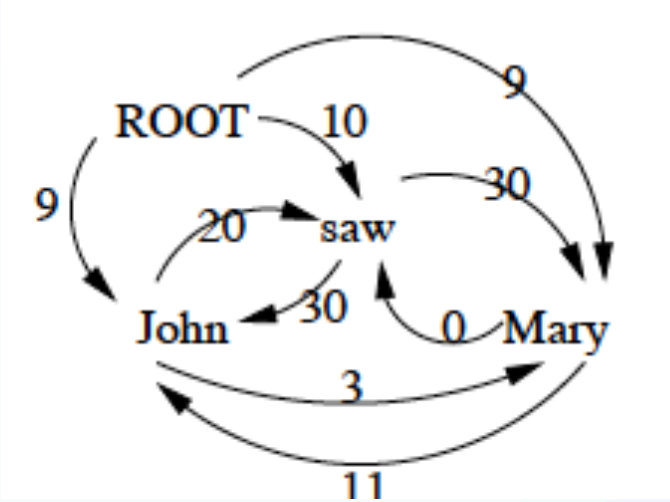
- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)
- Sketch of algorithm:
  - For each node, greedily select incoming arc with max w
  - If the resulting set of arcs forms a tree, this is the MST.
  - If not, there must be a cycle.
    - "Contract" the cycle: Treat it as a 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

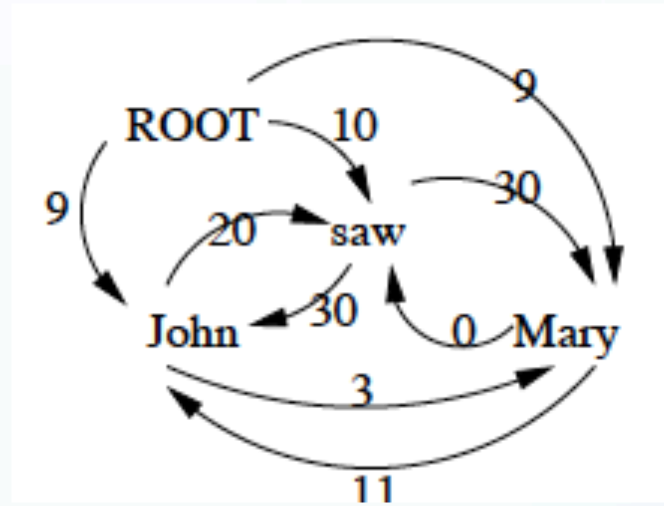
- Find maximum incoming arcs

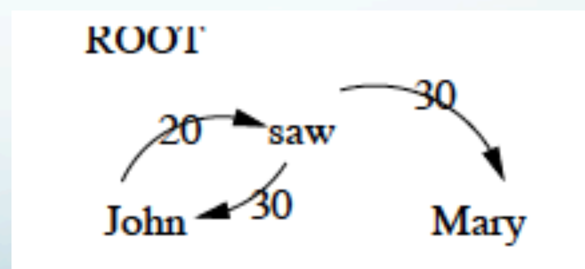


CLE: Step 1

- Find maximum incoming arcs

  - Is the result a tree?

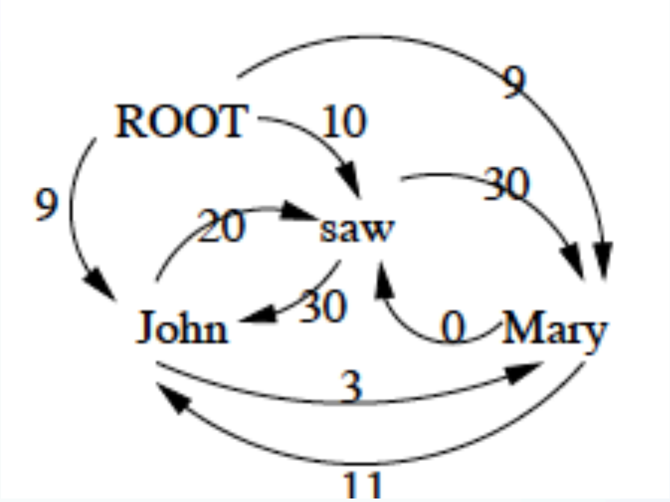


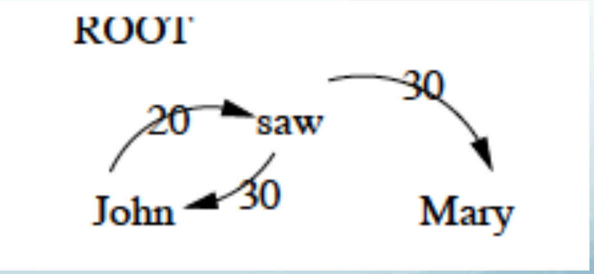


CLE: Step 1

- Find maximum incoming arcs

  - Is the result a tree?
    - No
  - Is there a cycle?

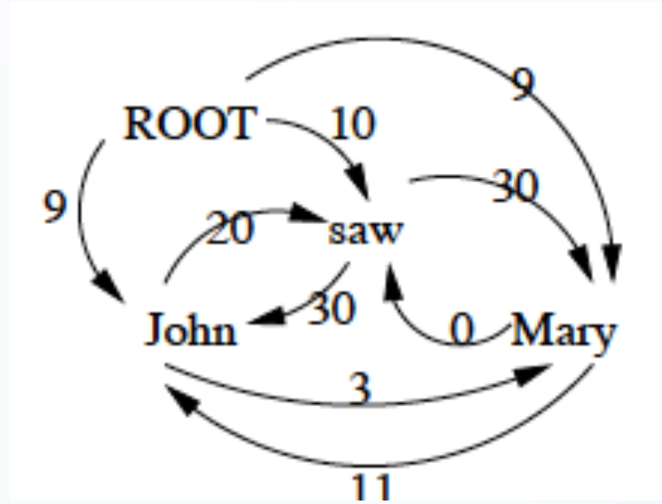


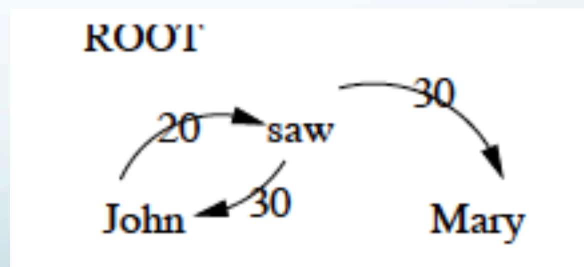


CLE: Step 1

- Find maximum incoming arcs

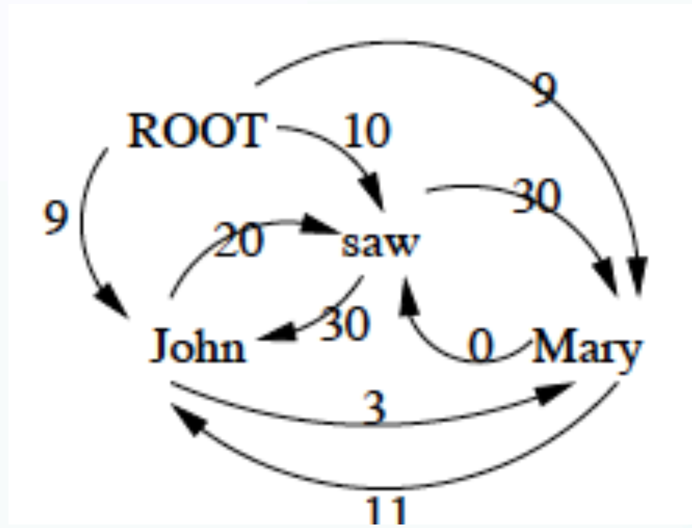
  - Is the result a tree?
    - No
  - Is there a cycle?
    - Yes, John/saw





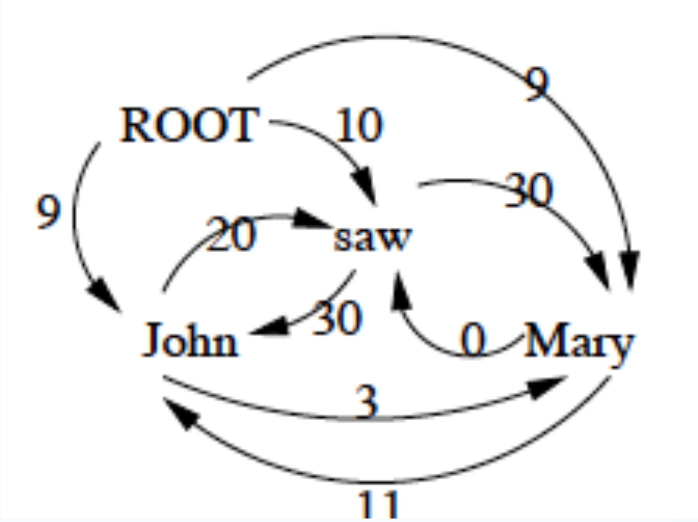
CLE: Step 2

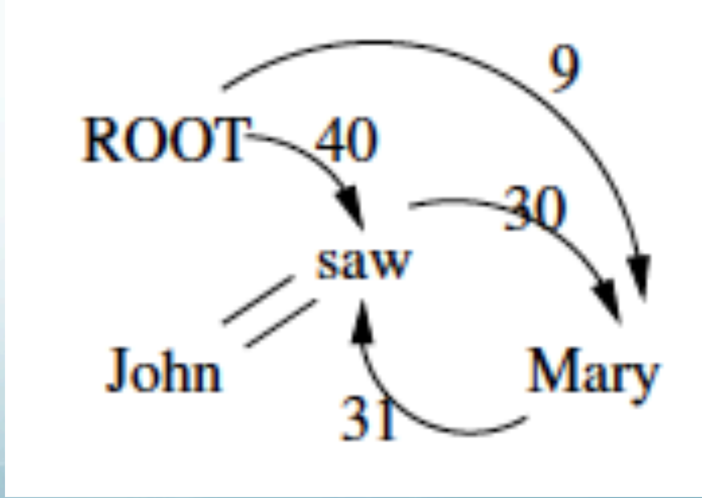
- Since there's a cycle:
  - Contract cycle & reweight
  - John+saw as single vertex



CLE: Step 2

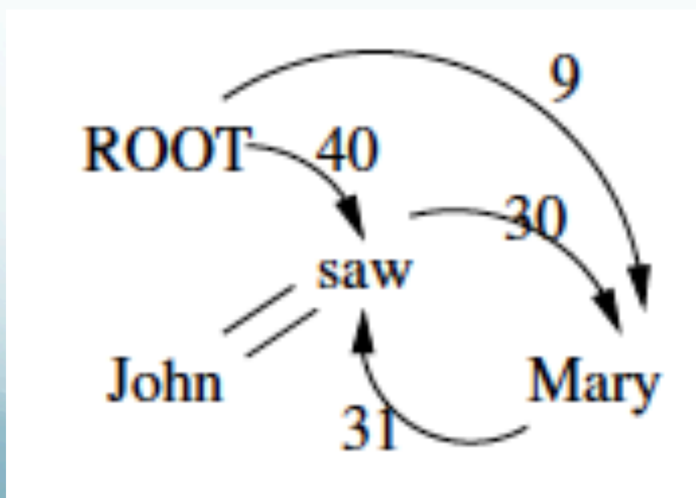
- Since there's a cycle:
  - Contract cycle & reweight
  - John+saw as single vertex
  - Calculate weights in & out as:
    - Maximum based on internal arcs and original nodes
    - Just single outside arc + (at most) inside
- Recurse





CLE: Recursive Step

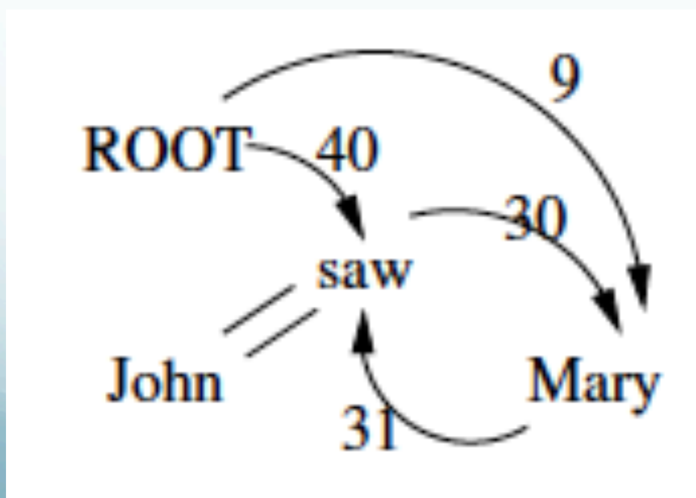
- In new graph, find graph of
  - Max weight incoming arc for each word

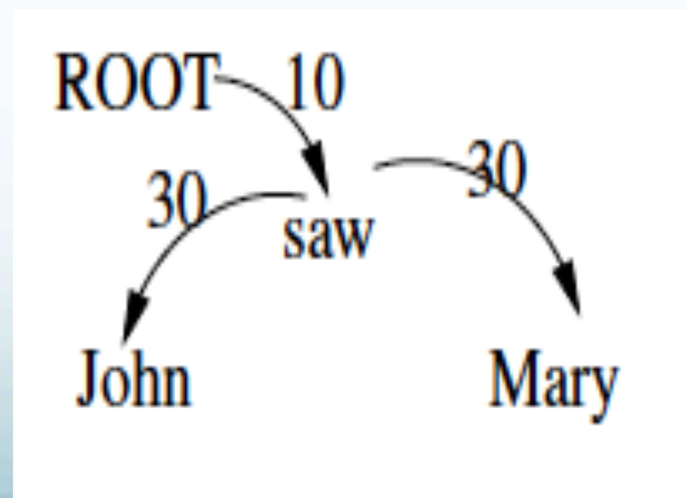


CLE: Recursive Step

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

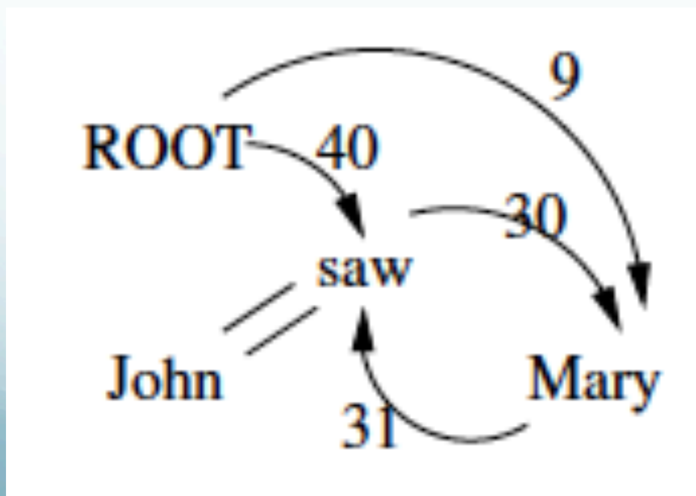
- Is it a tree?

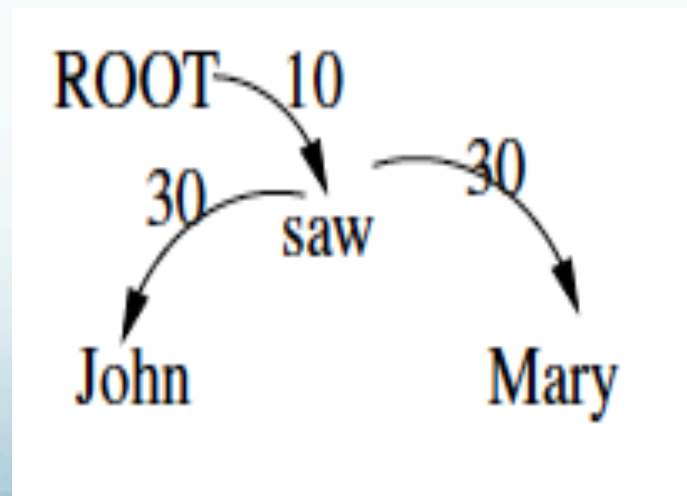




CLE: Recursive Step

- 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





Learning Weights

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

Learning Weights

- Weights for arc-factored model learned from corpus
  - Weights learned for tuple (w_i, w_j, l)
- 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, w_j, l)
- McDonald et al, 2005 employed discriminative ML
  - Perceptron algorithm or large margin variant
- Features: Local
  - Base features
    - Identity, POS of w_i, w_j; Label, direction of l
    - Sequence of POS tags, words between w_i, w_j
    - POS of words adjacent to w_i, w_j
  - Also conjunctions of features
    - Projective tree not required

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 $O(n^2)$
  - ...

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

$$\begin{bmatrix} \text{NUMBER} & \text{PL} \end{bmatrix}$$

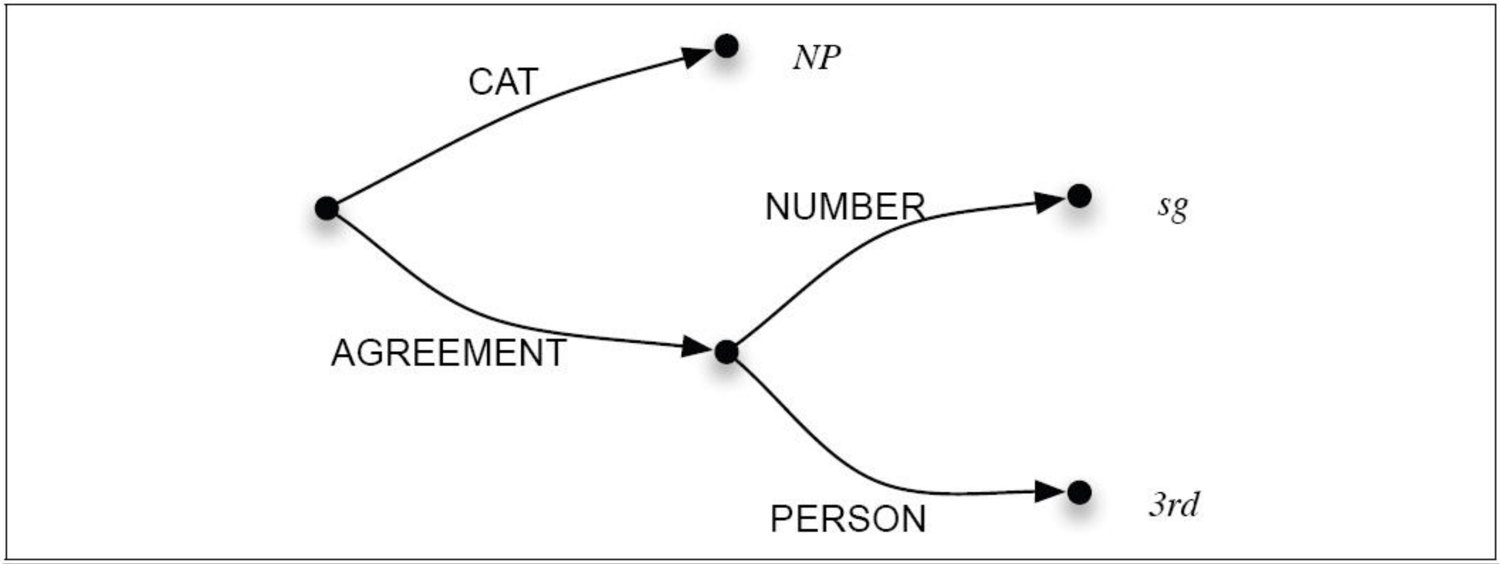
$$\begin{bmatrix} \text{PERSON} & 3 \end{bmatrix}$$

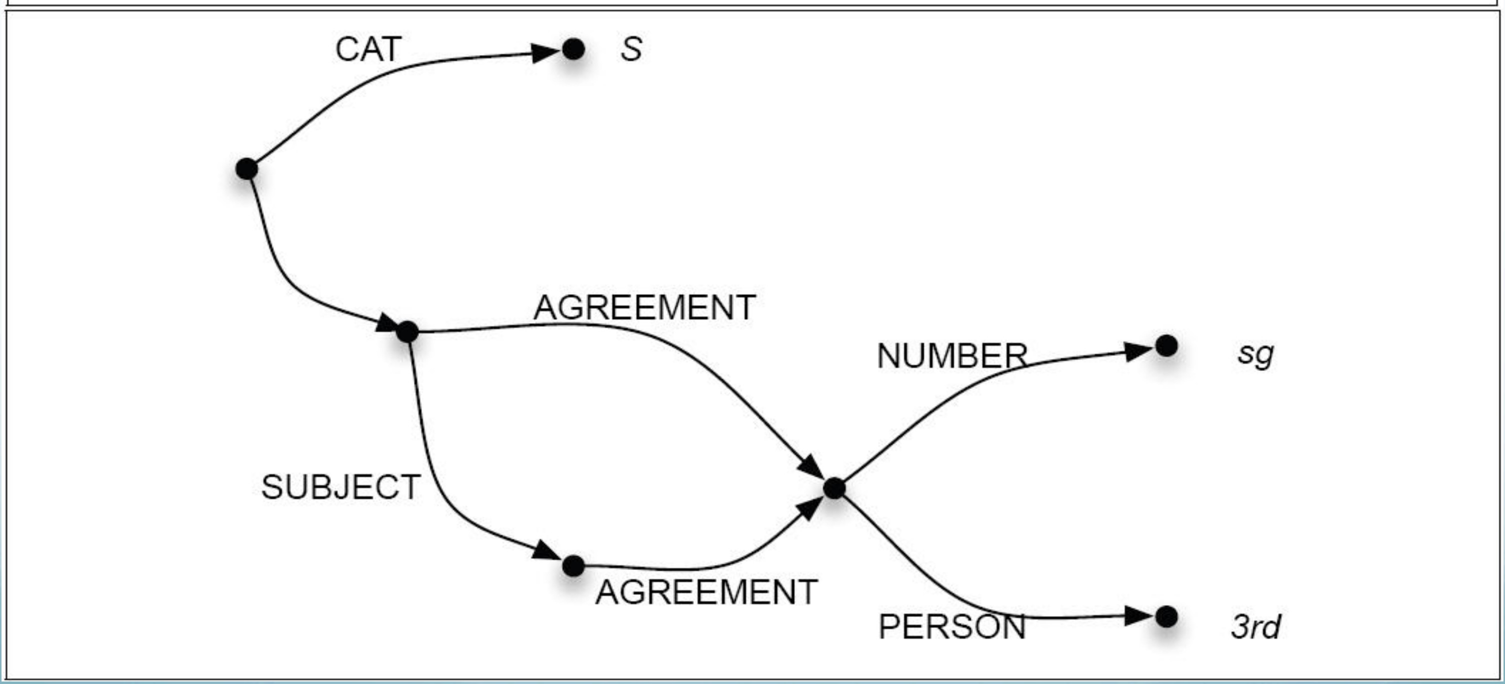
$$\begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{PERSON} & 3 \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUMBER} & \text{PL} \\ \text{PERSON} & 3 \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{HEAD} & \begin{bmatrix} \text{AGREEM'T} & \boxed{1} \begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$


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

$$\left[\begin{array}{ll} \text{AGREEMENT} & [1] \\ \text{SUBJECT} & \left[\begin{array}{ll} \text{AGREEMENT} & [1] \end{array}\right] \end{array}\right] \quad \text{U}$$

$$\left[\begin{array}{ll} \text{SUBJECT} & \left[\begin{array}{ll} \text{AGREEMENT} & \left[\begin{array}{ll} \text{PERSON} & 3 \\ \text{NUMBER} & \text{SG} \end{array}\right] \end{array}\right] \end{array}\right] \quad =$$

$$\left[\begin{array}{ll} \text{AGREEMENT} & [1] \\ \text{SUBJECT} & \left[\begin{array}{ll} \text{AGREEMENT} & [1] \left[\begin{array}{ll} \text{PERSON} & 3 \\ \text{NUMBER} & \text{SG} \end{array}\right] \end{array}\right] \end{array}\right]$$

Features in CFGs: Agreement

- Goal:
  - Support agreement of NP/VP, Det Nominal
- Approach:
  - Augment CFG rules with features
  - Employ head features
    - Each phrase: VP, NP has head
      - Head: child that provides features to phrase
        - Associates grammatical role with word
        - VP – V; NP – Nom, etc

Agreement with Heads and Features

VP -> Verb NP
<VP HEAD> = <Verb HEAD>

NP -> Det Nominal
<NP HEAD> = <Nominal HEAD>
<Det HEAD AGREEMENT> = <Nominal HEAD AGREEMENT>

Nominal -> Noun
<Nominal HEAD> = <Noun HEAD>

Noun -> flights
<Noun HEAD AGREEMENT NUMBER> = PL

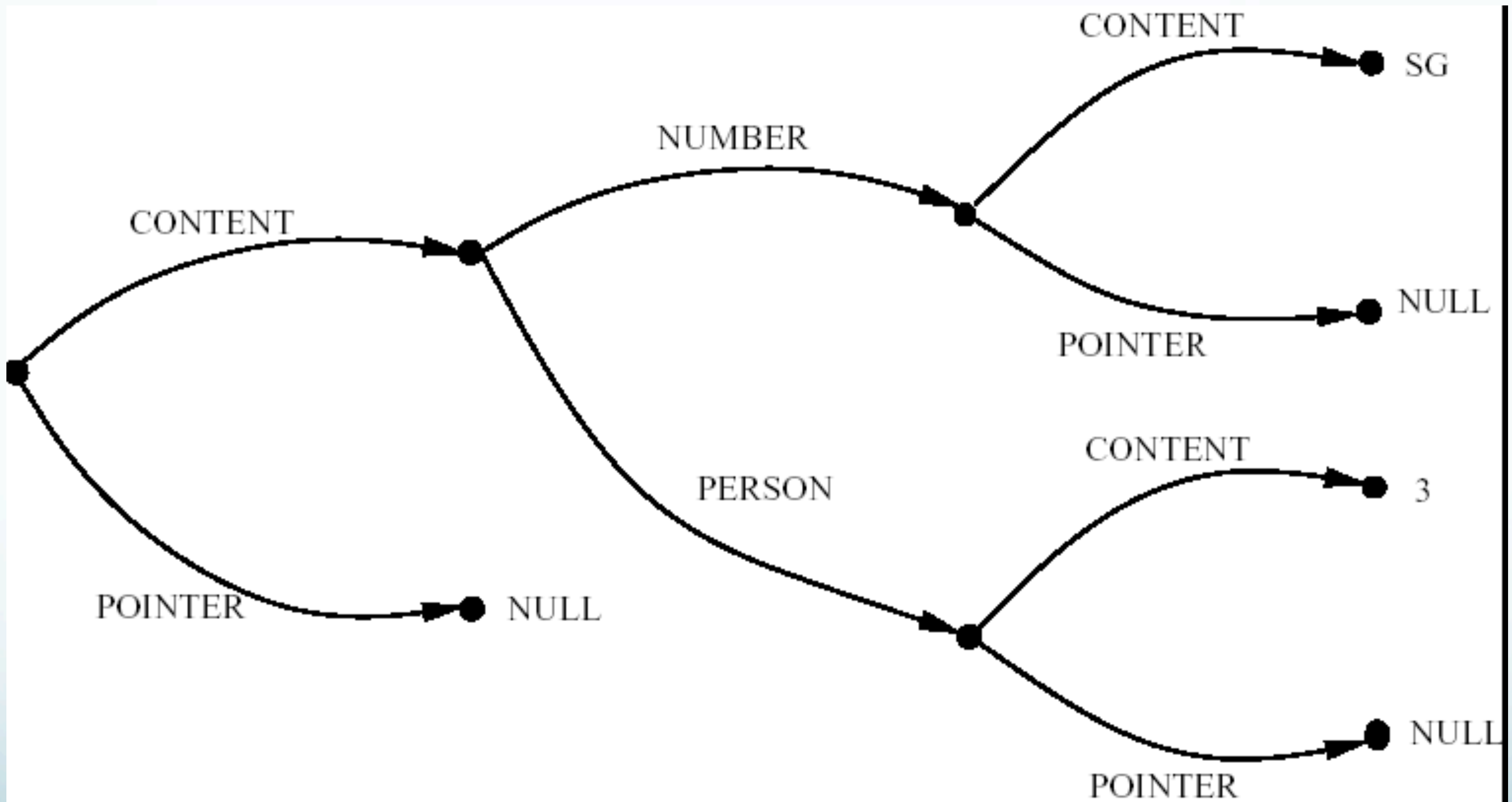
Verb -> serves
<Verb HEAD AGREEMENT NUMBER> = SG
<Verb HEAD AGREEMENT PERSON> = 3

Feature Applications

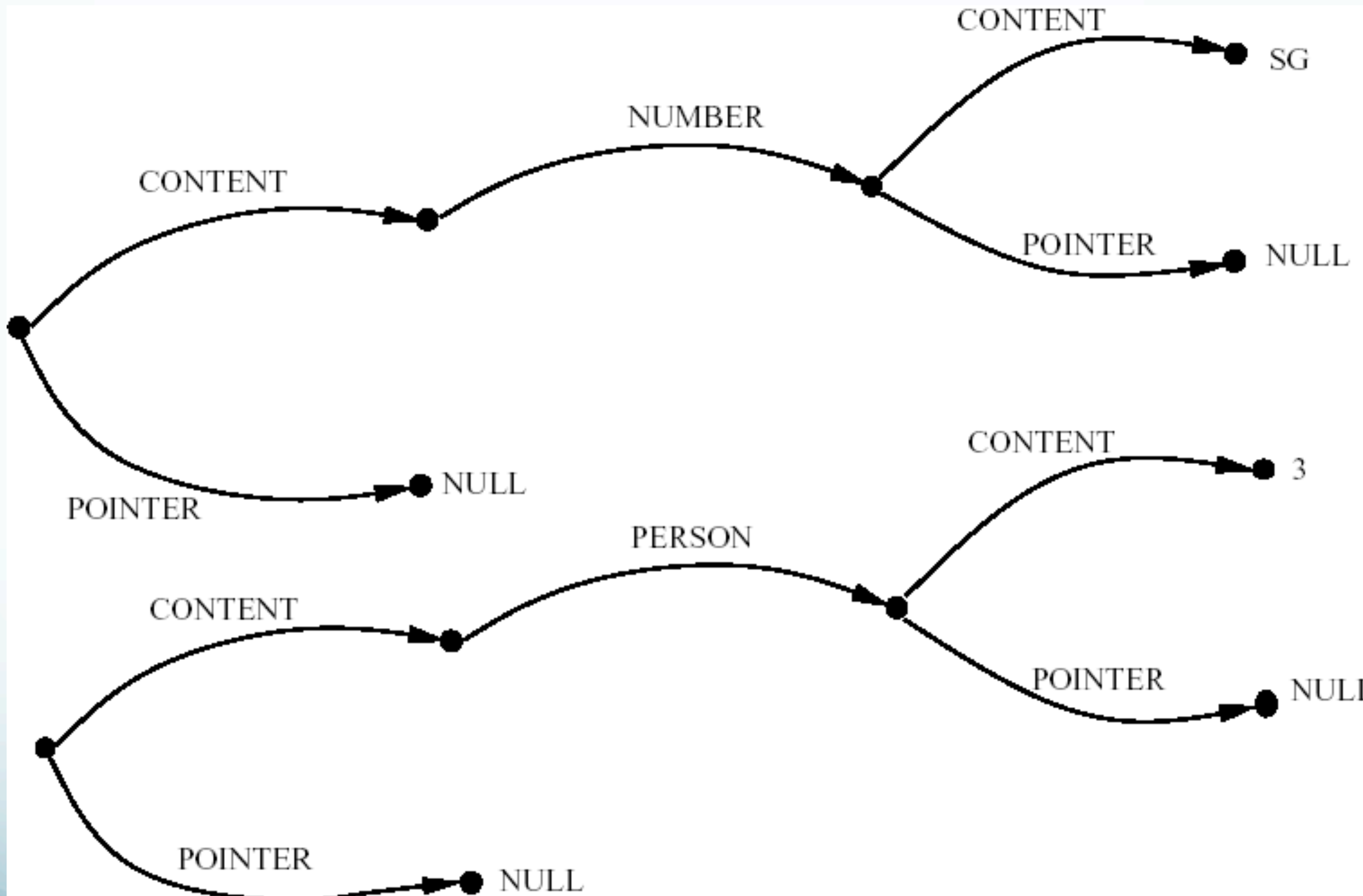
- Subcategorization:
  - Verb-Argument constraints
    - Number, type, characteristics of args (e.g. animate)
    - Also adjectives, nouns
- Long distance dependencies
  - E.g. filler-gap relations in wh-questions, rel

Implementing Unification

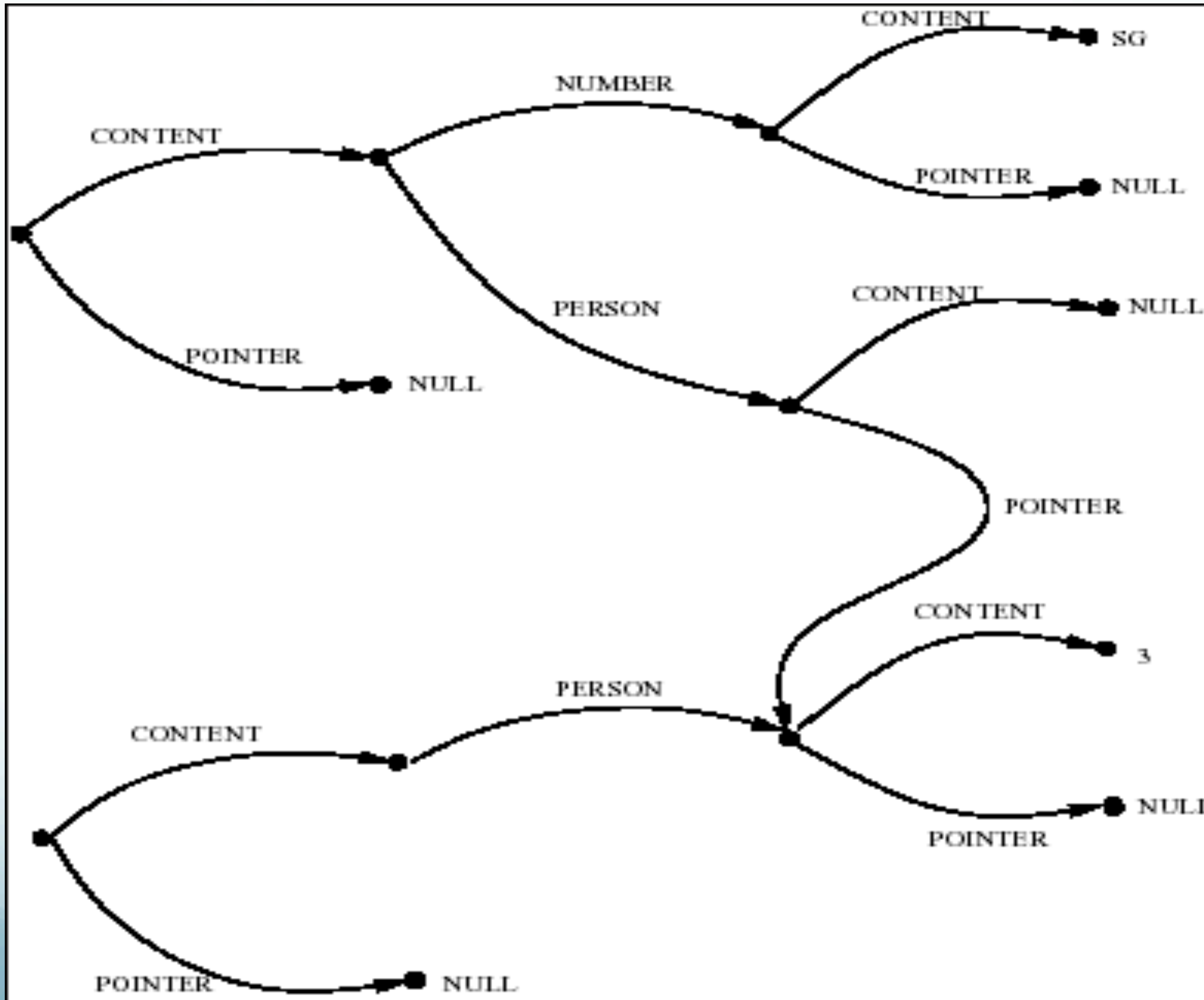
- Data Structure:
  - Extension of the DAG representation
  - Each f.s. has a content field and a pointer field
    - If pointer field is null, content field has the f.s.
    - If pointer field is non-null, it points to actual f.s.

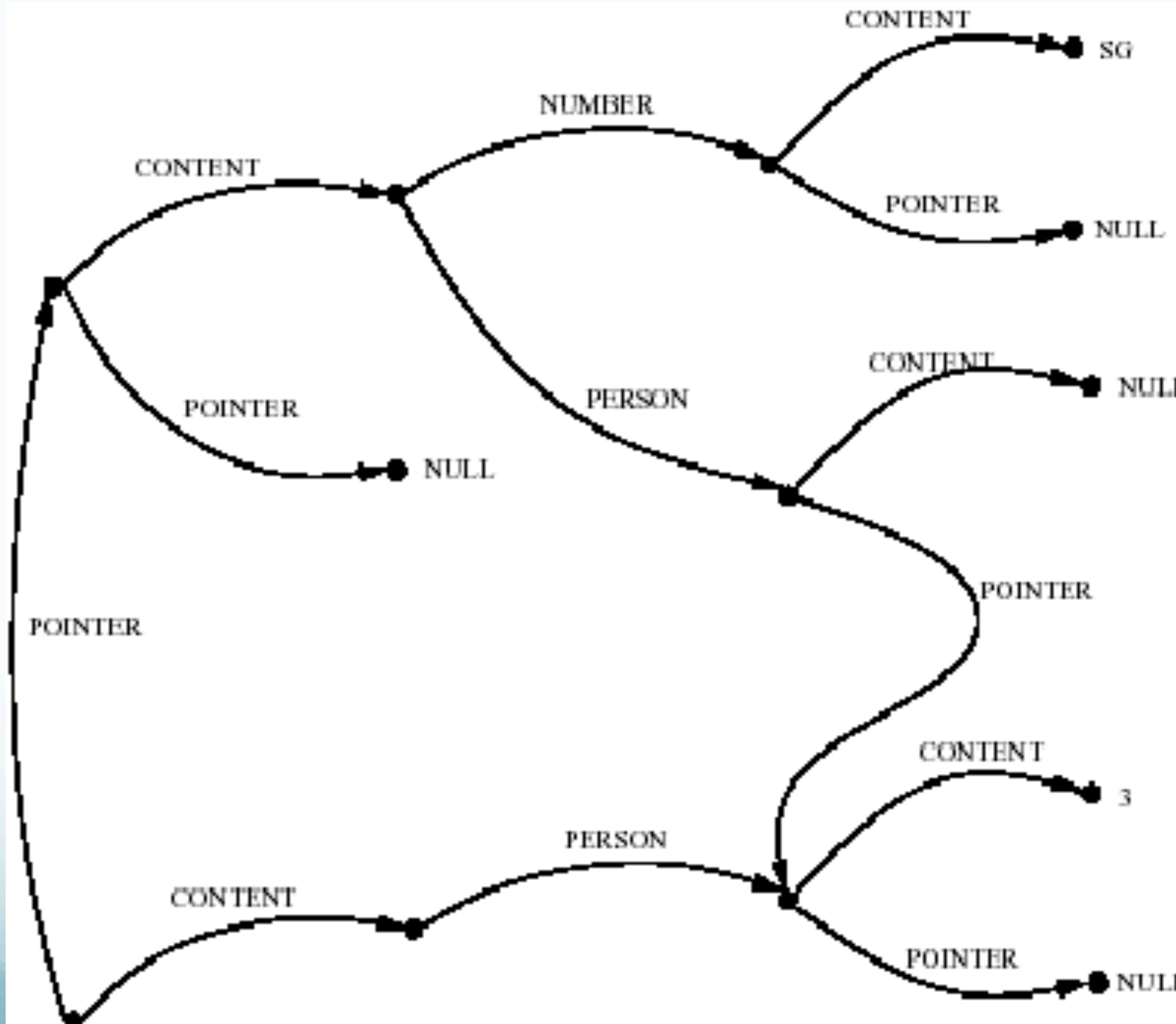


NUMBER	SG
PERSON	3



CONTENT
SG
NUMBER
CONTENT
POINTER
NULL
POINTER
NULL
CONTENT
3
PERSON
CONTENT
POINTER
NULL
POINTER
NULL


CONTENT
SG
NUMBER
CONTENT
POINTER
NULL
PERSON
CONTENT
NULL
POINTER
NULL
POINTER
CONTENT
3
PERSON
CONTENT
NULL
POINTER
POINTER
NULL


CONTENT
SG
NUMBER
CONTENT
POINTER
NULL
CONTENT
NULL
POINTER
NULL
PERSON
POINTER
POINTER
CONTENT
3
PERSON
CONTENT
POINTER
NULL

Implementing Unification: II

- Algorithm:
  - Operates on pairs of feature structures
    - Order independent, destructive
  - If fs1 is null, point to fs2
  - If fs2 is null, point to fs1
  - If both are identical, point fs1 to fs2, return fs2
    - Subsequent updates will update both
  - If non-identical atomic values, fail!

Implementing Unification: III

- If non-identical, complex structures
  - Recursively traverse all features of fs2
  - If feature in fs2 is missing in fs1
    - Add to fs1 with value null
  - If all unify, point fs2 to fs1 and return fs1

Example

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

$$\begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

[AGREEMENT [1]] U [AGREEMENT [PERSON 3]]

[NUMBER SG] U [PERSON 3]

[NUMBER SG] U [PERSON 3]
[PERSON NULL]

Unification and the Earley Parser

- Employ constraints to restrict addition to chart
- Actually pretty straightforward
  -

Unification and the Earley Parser

- Employ constraints to restrict addition to chart
- Actually pretty straightforward
  - Augment rules with feature structure

Unification and the Earley Parser

- Employ constraints to restrict addition to chart
- Actually pretty straightforward
  - Augment rules with feature structure
  - Augment state (chart entries) with DAG
    - Prediction adds DAG from rule
    - Completion applies unification (on copies)
      - Adds entry only if current DAG is NOT subsumed

Unification Parsing

- Abstracts over categories
  - S-> NP VP =>
    - X0 -> X1 X2; <X0 cat> = S; <X1 cat>=NP;
    - <X2 cat>=VP
  - Conjunction:
    - X0->X1 and X2; <X1 cat> =<X2 cat>;
    - <X0 cat>=<X1 cat>
- Issue: Completer depends on categories
- Solution: Completer looks for DAGs which unify with the just-completed state's DAG

Extensions

- Types and inheritance
  - Issue: generalization across feature structures
    - E.g. many variants of agreement
      - More or less specific: 3rd vs sg vs 3rdsg
  - Approach: Type hierarchy
    - Simple atomic types match literally
    - Multiple inheritance hierarchy
      - Unification of subtypes is most general type that is more specific than two input types
    - Complex types encode legal features, etc

Conclusion

- Features allow encoding of constraints
  - Enables compact representation of rules
  - Supports natural generalizations
- Unification ensures compatibility of features
  - Integrates easily with existing parsing mech.
- Many unification-based grammatical theories