Dependency Parsing & Feature-based Parsing

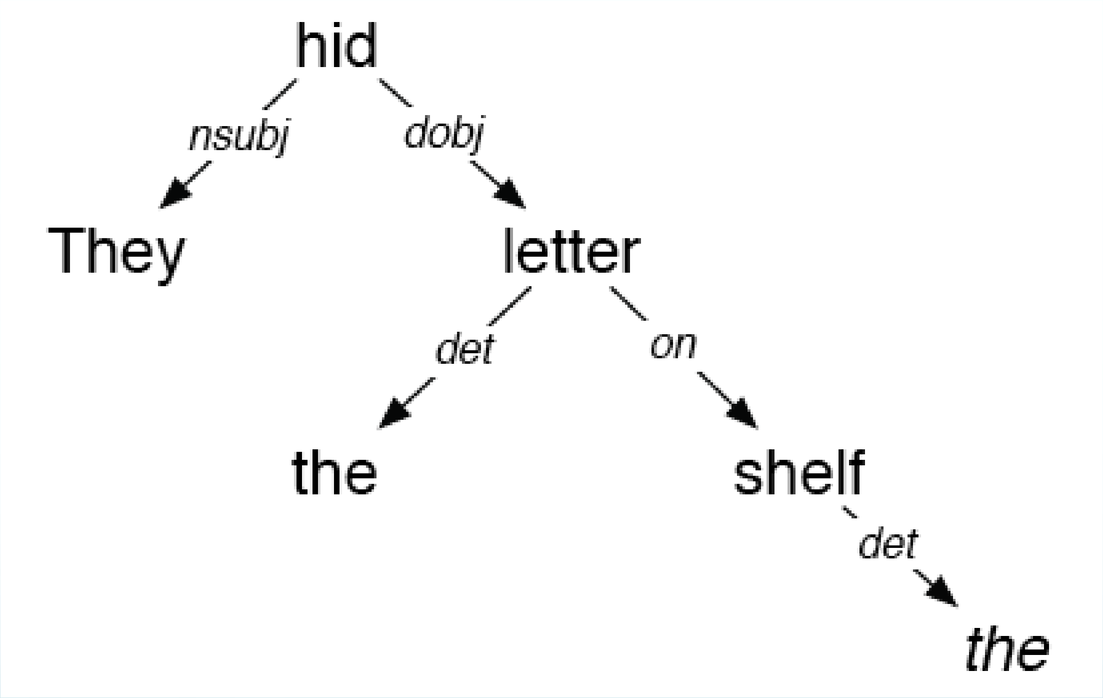
Ling571
Deep Processing Techniques for NLP
February 1, 2017

Roadmap

- Dependency parsing
  - Transition-based parsing
    - Configurations and Oracles
- Feature-based parsing
  - Motivation
  - Features
  - Unification

Dependency Parse Example

- They hid the letter on the shelf



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

Nivre's Arc-Standard Oracle

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

Nivre's Arc-Standard Oracle

- Words: $w_1,\ldots,w_n$; 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)

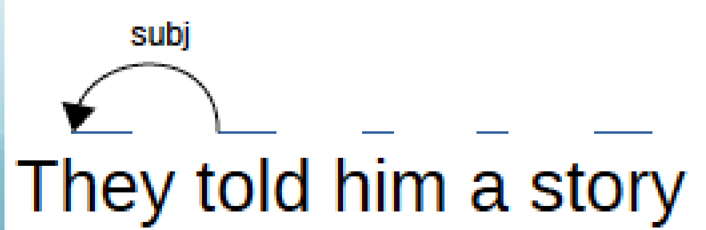
Action	Stack	Buffer
	[]	[They told him a story]
Shift	[They]	[told him a story]

Example (Ballesteros '15)

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

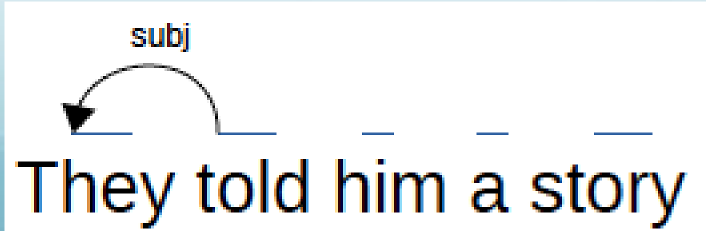
Example (Ballesteros '15)

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)



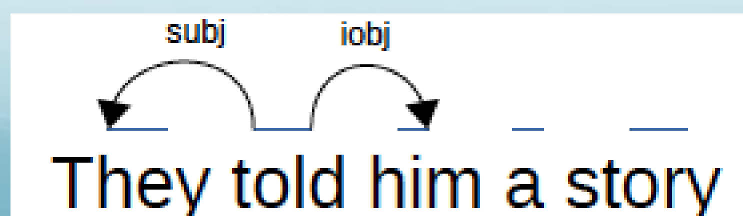
Example (Ballesteros '15)

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]



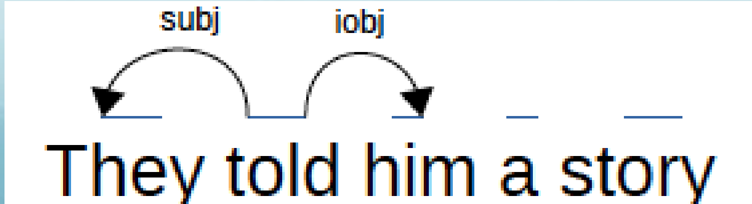
Example (Ballesteros '15)

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]



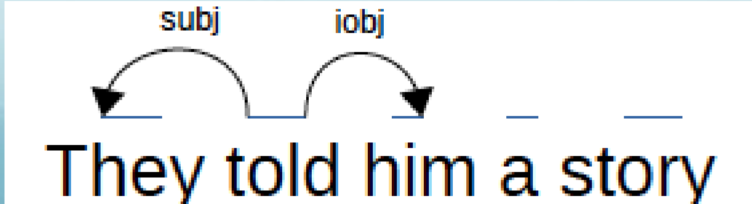
Example (Ballesteros '15)

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]



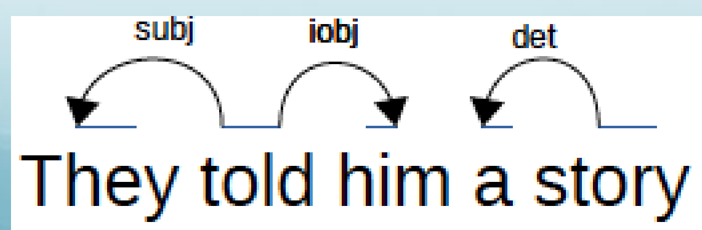
Example (Ballesteros '15)

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]	[]



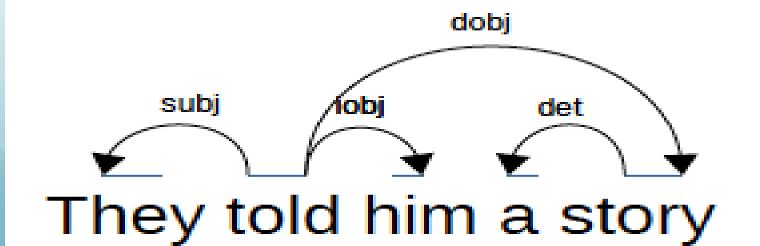
Example (Ballesteros ‘15)

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]	[]



- Is this a parser?

- Is this a parser?
- No!
  - Uses known dependency tree information
- Creates transition-based representation of tree/parse
  - I.e. the sequence of transitions that derive it

Transition-Based Parsing

- Find best sequence of transitions given input, model

Transition-Based Parsing

- Find best sequence of transitions given input, model
  - Can be viewed as finding highest scoring sequence
    - E.g. SH, LA, SH, RA, SH, SH, LA, RA
  - Where sequence score is sum of transition scores

Transition-Based Parsing

- Find best sequence of transitions given input, model
  - Can be viewed as finding highest scoring sequence
    - E.g. SH, LA, SH, RA, SH, SH, LA, RA
  - Where sequence score is sum of transition scores
  - Aka deterministic transition-based parsing
- Greedy approach: O(n) in length of input
  - Assuming O(1) access to highest scoring transition

Greedy Transition-Based Parsing

- Parser ($x = (w_1, \ldots, w_n)$)
  - Initialize configuration c to start configuration on x
  - While not in terminal configuration:
    - Find best transition t^* on configuration c
    - Update c based on t^*
  - Return set of output arcs (A)

- How do we pick the best transition?

Transition Classification

- How do we pick the best transition?
- Train classifier to predict transitions:
  - {Left-Arc,Right-Arc} x Number of dependency labels

Transition Classification

- How do we pick the best transition?
- 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

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

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
- State-of-art dependency parsing:
  - UAS: 92.5%; LAS: 90.3%

Building Transition-based Parser

- Given a dependency treebank
  - Use oracle algorithm to create transition sequences
  - Train classifier on transition x configuration features
  - Build greedy algorithm based on classifications
  - Evaluate on new sentences

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 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

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

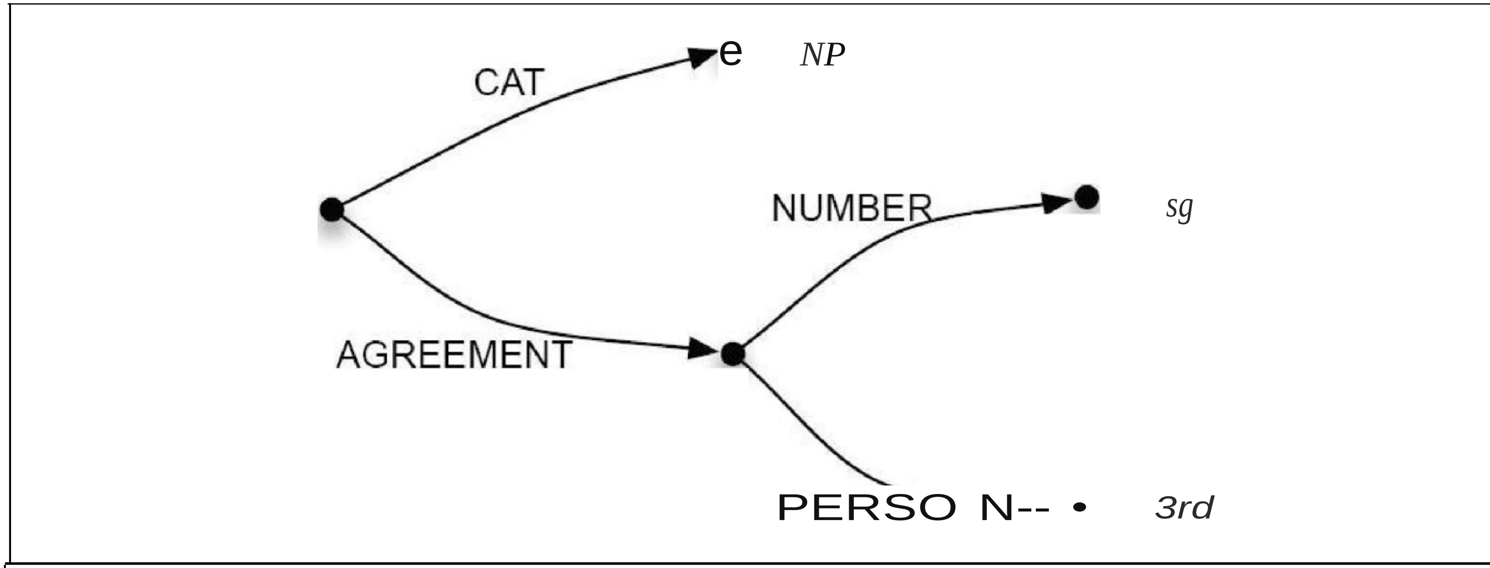
$$\begin{bmatrix} \text{PERSON} & 3 \end{bmatrix}$$

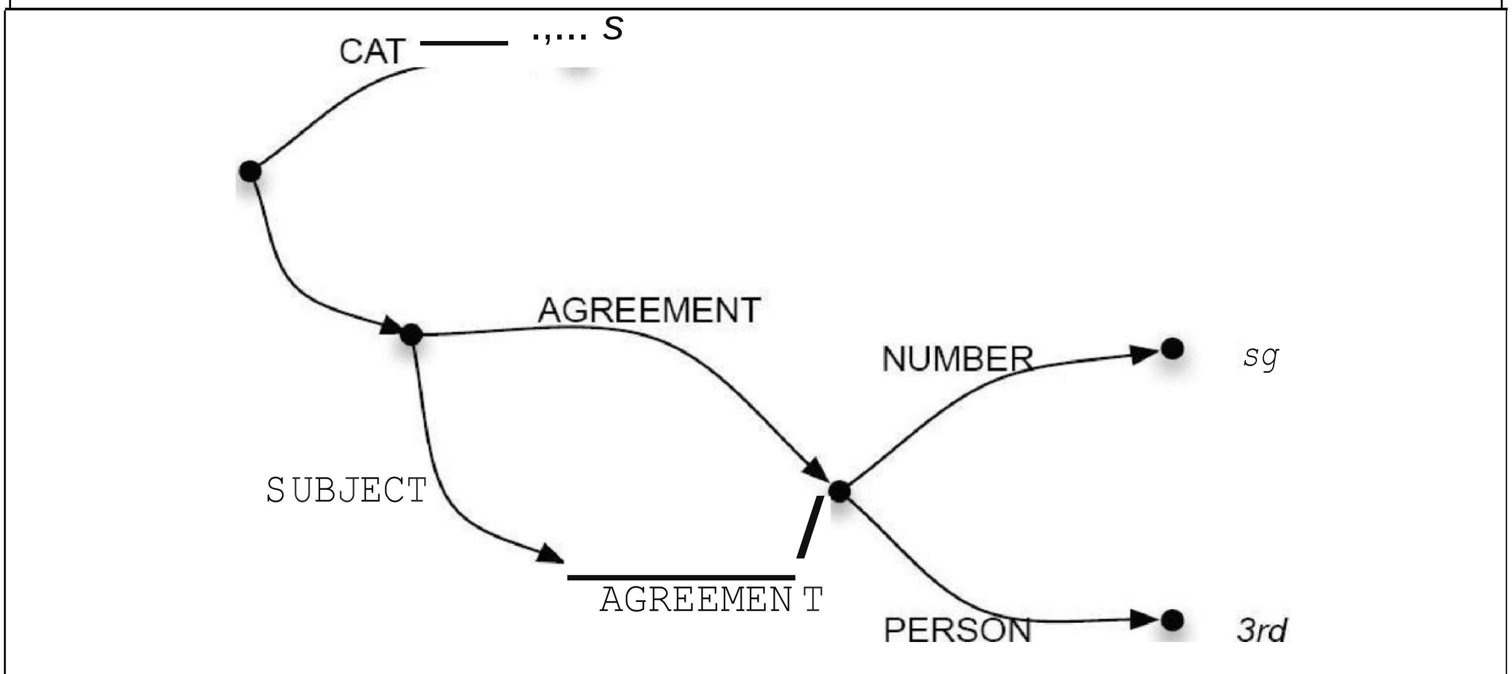
$$\begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{PERSON} & 3 \end{bmatrix}$$

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

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

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


CAT
NP
NUMBER
sg
AGREEMENT
PERSO N
3rd


CAT
S
AGREEMENT
NUMBER
sg
SUBJECT
AGREEMEN T
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

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

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

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

Unification and Parsing

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

Unification and Parsing

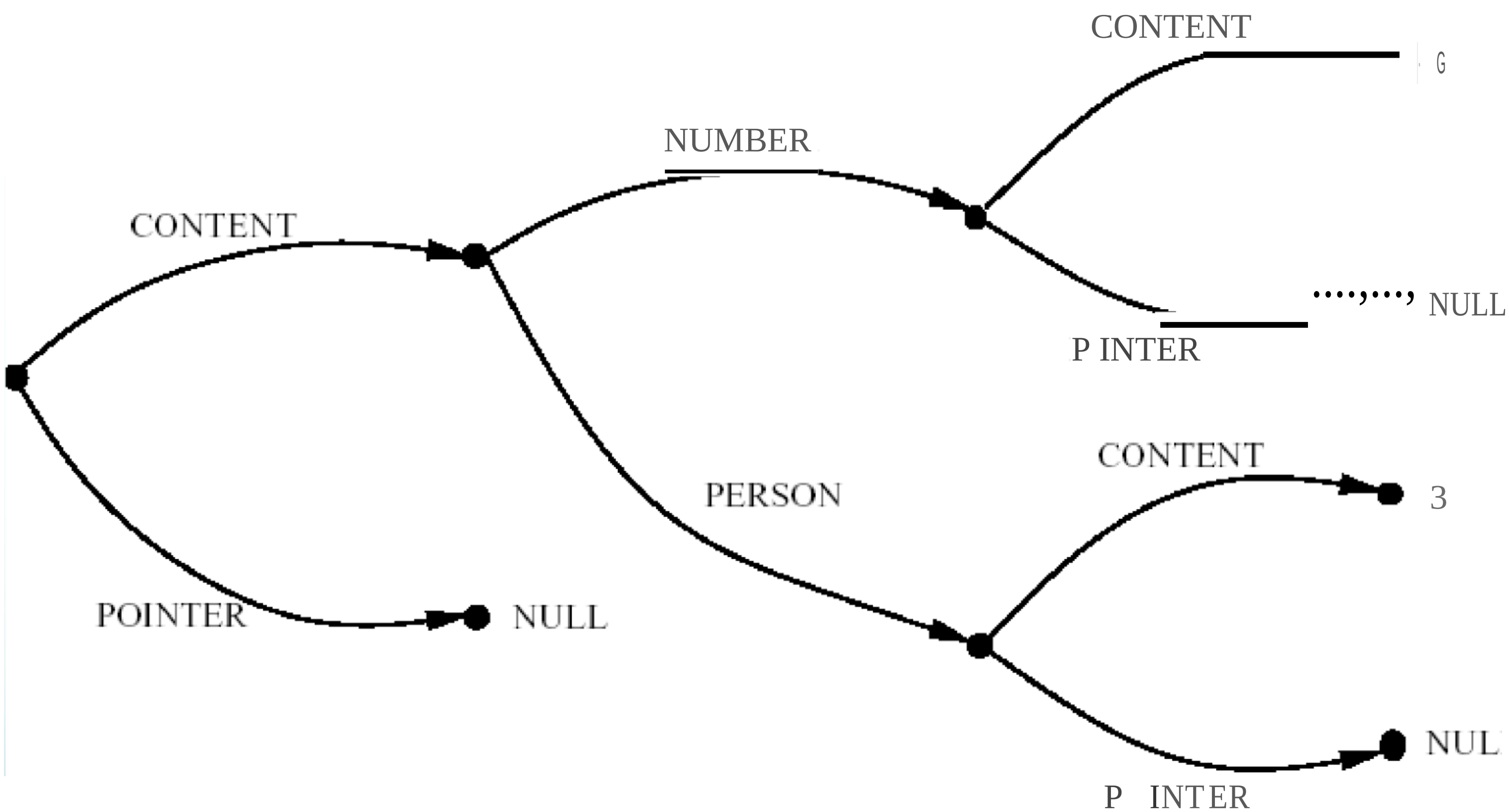
- Employ constraints to restrict addition to chart
- Actually pretty straightforward
  - Augment rules with feature structure

Unification and Parsing

- 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

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

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

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

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

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

[NUMBER SG] U [PERSON 3]

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

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.

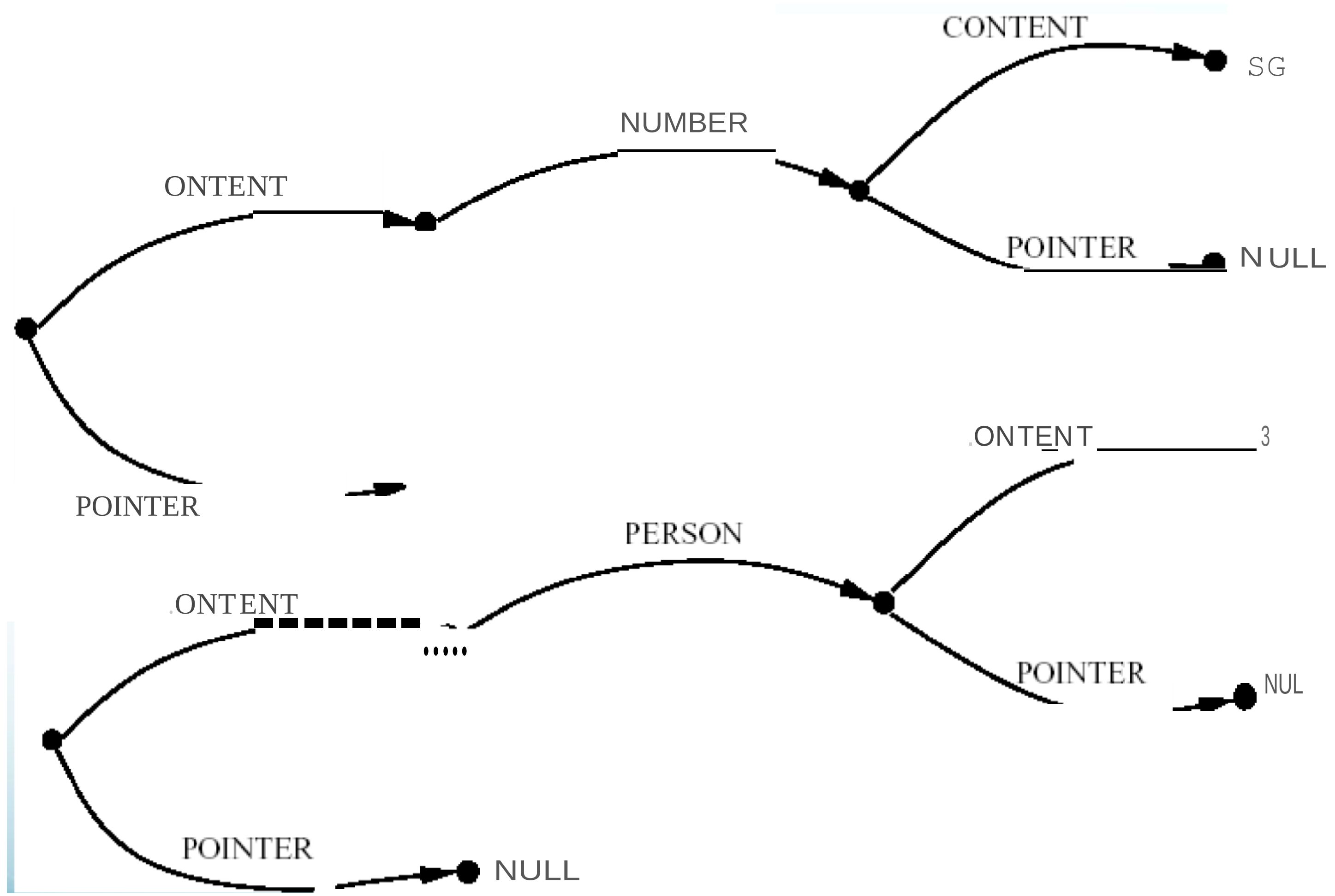
- Many unification-based grammatical theories

Unification Parsing

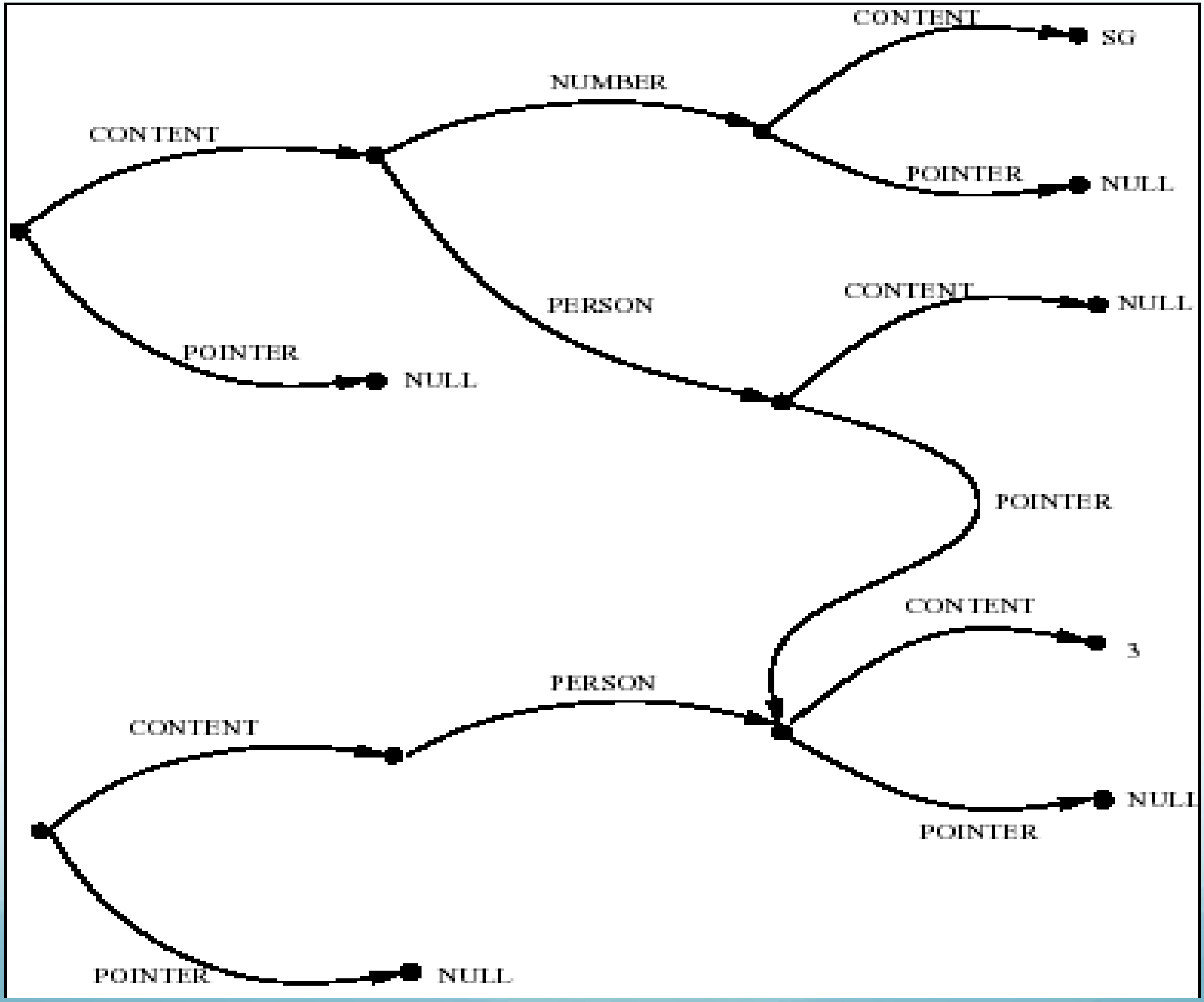
- Abstracts over categories
  - S-> NP VP =>
    - X0 -> X1 X2; <X0 cat> = S; <X1 cat>=NP;
    - <X2 cat>=VP
  - Conjunction:
    - X0->X1 and X2; <X1 cat> =<X2 cat>;
    - <X0 cat>=<X1 cat>
- Issue: Completer depends on categories
- Solution: Completer looks for DAGs which unify with the just-completed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

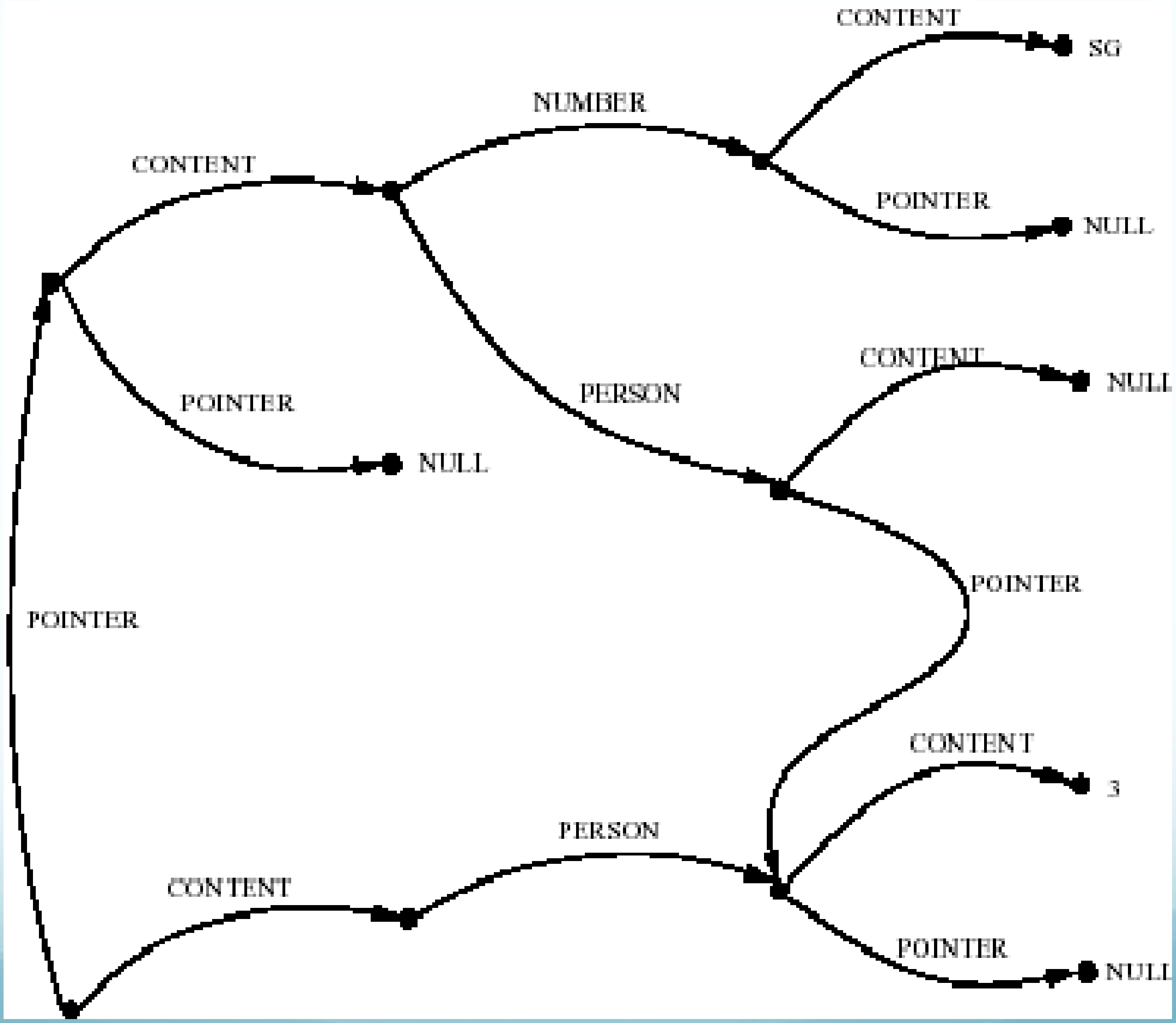
Extensions

- 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



CONTENT
SG
NUMBER
ONTENT
POINTER
NULL
.ONTENT
3
POINTER
PERSON
.ONTENT
POINTER
NUL
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
PERSON
POINTER
NULL
POINTER
POINTER
CONTENT
3
PERSON
CONTENT
POINTER
NULL