CS114 Lecture 14

Dependency Grammars and Functional Unification Grammars

March 11, 2013

Professor Meteer

Slides from UPenn, Adapted from slides by Kathy McCoy, University of Delaware

Another Earley Example

Spec → S
S → NP VP
S → VP
NP → Det Noun
NP → PrN
VP → V
VP → V NP
Det → a | the
N → book
V → Mark | read
PrN → Mark

CHART 1

S0	Spec → S		
S1	S → . NP VP	[0,0]	Predictor
S2	S → . VP	[0,0]	Predictor
S3	NP → . Det Noun	[0,0]	Predictor
S4	NP → . PrN	[0,0]	Predictor
S5	Det → . A	[0,0]	Predictor
S6	Det → . the	[0,0]	Predictor
S7	PrN → . Mark	[0,0]	Predictor
S8	VP → . V	[0,0]	Predictor
S9	VP → . V NP	[0,0]	Predictor
S10	V → . Mark	[0,0]	Predictor
S11	V → . read	[0,0]	Predictor

Chart 1: Mark read

CHART 1

S0	Spec → S		
S1	S → . NP VP	[0,0]	
S2	S → . VP	[0,0]	
S3	NP → . Det Noun	[0,0]	
S4	NP → . PrN	[0,0]	
S5	Det → . A	[0,0]	
S6	Det → . the	[0,0]	
S7	PrN→ . Mark	[0,0]	
S8	VP → . V	[0,0]	
S9	VP → . V NP	[0,0]	
S10	V→ . Mark	[0,0]	
S11	V→ . read	[0,0]	

S12 PrN→ Mark .	[0,1]	Scanner	S7
S13 V → Mark .	[0,1]	Scanner	S10
S14 VP → V .	[0,1]	Completer	S13
S15 S → VP .	[0,1]	Completer	S14
S16 Spec → S .	[0,1]	Completer	S15
S17VP → V . NP	[0,1]	Completer	S13
S18NP → PrN .	[0,1]	Completer	S12
S19S → NP . VP	[0,1]	Completer	S18
S20 NP → . Det Noun	[1,1]	Predictor S17	
S21 NP → . PrN	[1,1]	Predictor S17	
S22 Det → . a	[1,1]	Predictor S20	
S23 Det → . the	[1,1]	Predictor S20	
S24 PrN→ . Mark	[1,1]	Predictor S21	
S25 VP → . V	[1,1]	Predictor S19	
S26 VP → . V NP	[1,1]	Predictor S19	
S27 V→ . Mark	[1,1]	Predictor S25,26	
S28 V→ . read	[1,1]	Predictor S25,26	

Chart 2: Mark Read

Chart 2

S29 V→ read .	[1,2]Scanner	
S30 VP → V .	[1,2]Completer	S29
S31 VP → V . NP	[1,2]Completer	S29
S32 S → VP .	[1,2]Completer	S30
S33 Spec → S .	[1,2]Completer	S32
S34 S → NP VP .	[0,2]Completer	S19, S30
S35 Spec → S.	**[0,2] Completer**	**S34**

S35 Spec → S .	**[1,2] Completer**	**S34**
S34 S → NP VP .	[0,2]Completer	S19, S30

S30 VP → V .	[1,2]Completer	S29
S29 V→ read .	[1,2]Scanner	

S19 S → NP . VP	[0,1]Completer	S18
S18 NP → PrN .	[0,1]Completer	S12
S12 PrN→ Mark .	[0,1]Scanner	S7

Dependency Grammars

- In CFG-style phrase-structure grammars the main focus is on *constituents.*
- But it turns out you can get a lot done with just binary relations among the words in an utterance.
- In a dependency grammar framework, a parse is a tree where
  - the nodes stand for the words in an utterance
  - The links between the words represent dependency relations between pairs of words.
    - Relations may be typed (labeled), or not.

Well-formedness

- A dependency graph is well-formed iff
  - Single head: Each word has only one head.
  - Acyclic: The graph should be acyclic.
  - Connected: The graph should be a single tree with all the words in the sentence.
  - Projective: If word A depends on word B, then all words between A and B are also subordinate to B (i.e. dominated by B).

Comparison

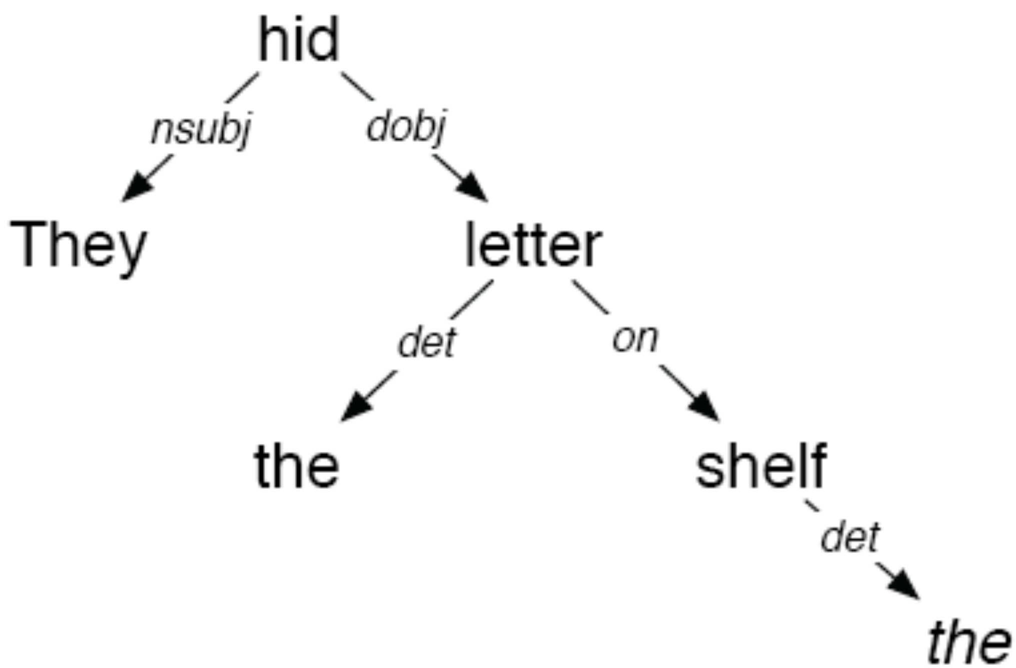
- Dependency structures explicitly represent
  - Head-dependent relations (directed arcs)
  - Functional categories (arc labels)
  - Possibly some structural categories (parts-of-speech)

- Phrase structure explicitly represent
  - Phrases (non-terminal nodes)
  - Structural categories (non-terminal labels)
  - Possibly some functional categories (grammatical functions)

Dependency Relations

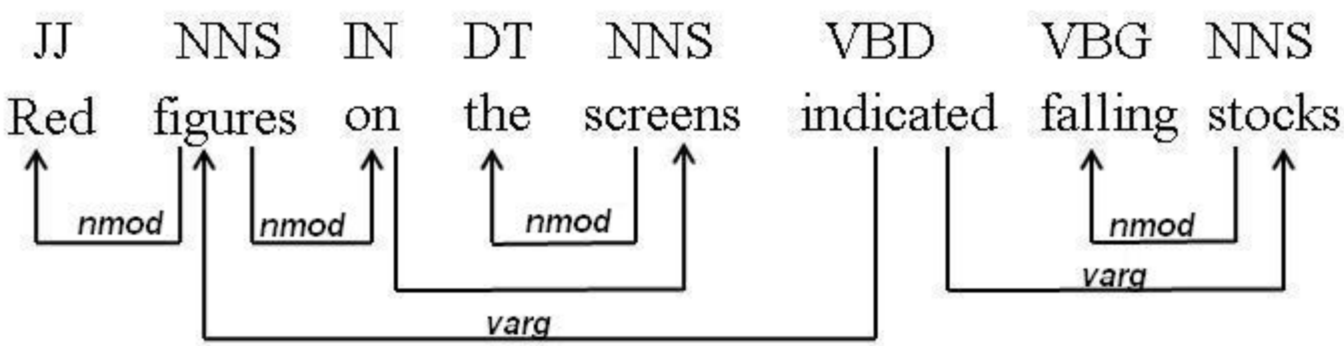
Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Dependency Parse



They hid the letter on the shelf

Dependency Tree with Labels



Dependency Parsing

- The dependency approach has a number of advantages over full phrase-structure parsing.
  - Deals well with free word order languages where the constituent structure is quite fluid
  - Parsing is much faster than CFG-bases parsers
  - Dependency structure often captures the syntactic relations needed by later applications
    - CFG-based approaches often extract this same information from trees anyway.

Dependency Parsing

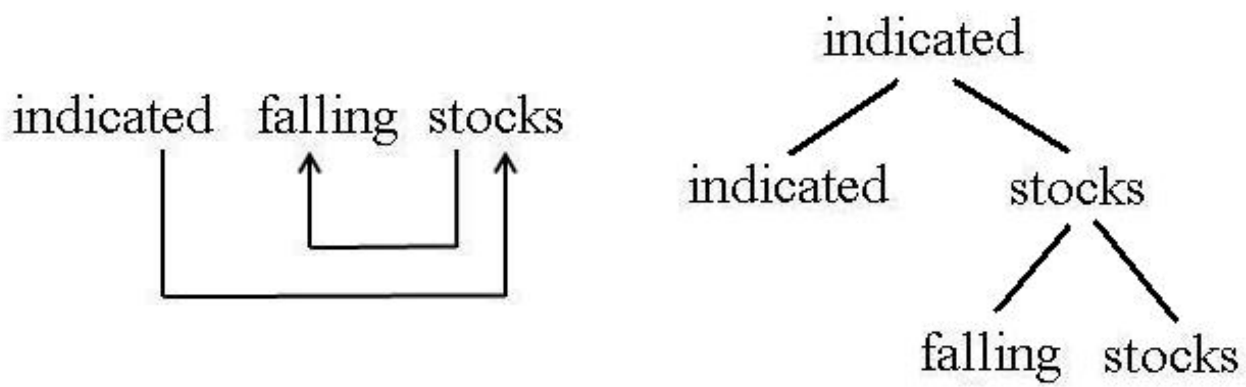
- There are two modern approaches to dependency parsing
  - Optimization-based approaches that search a space of trees for the tree that *best* matches some criteria
  - Shift-reduce approaches that greedily take actions based on the current word and state.

Parsing Methods

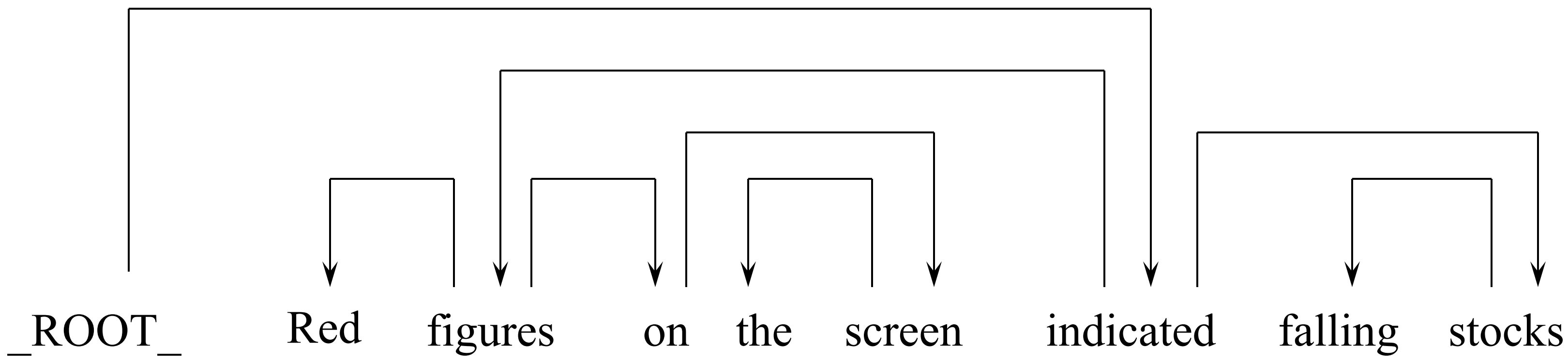
- Three main traditions
  - Dynamic programming
    - CYK, Eisner, McDonald
  - Constraint satisfaction
    - Maruyama, Foth et al., Duchier
  - Deterministic search
    - Covington, Yamada and Matsumuto, Nivre

Dynamic Programming

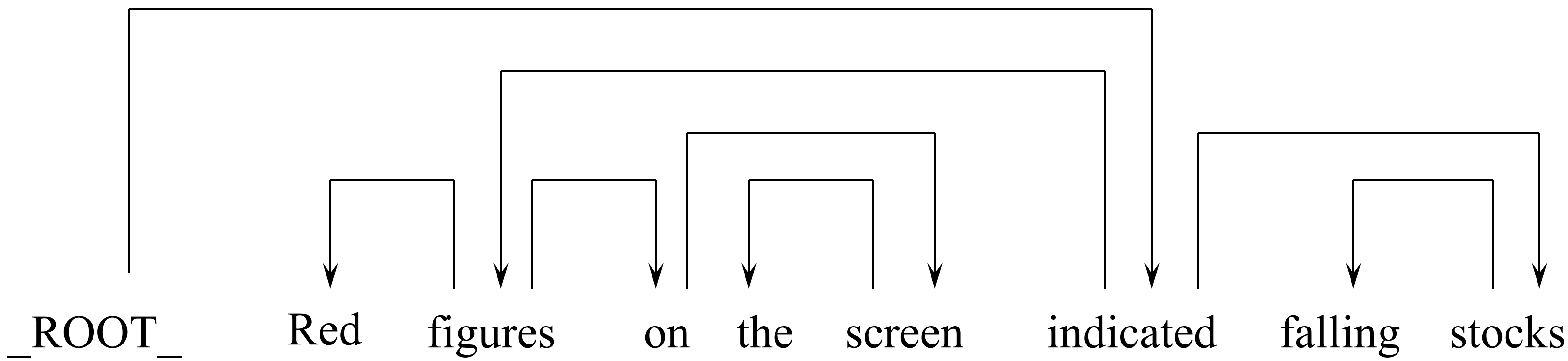
- Basic Idea: Treat dependencies as constituents.
- Use, e.g. , CYK parser (with minor modifications)



Example

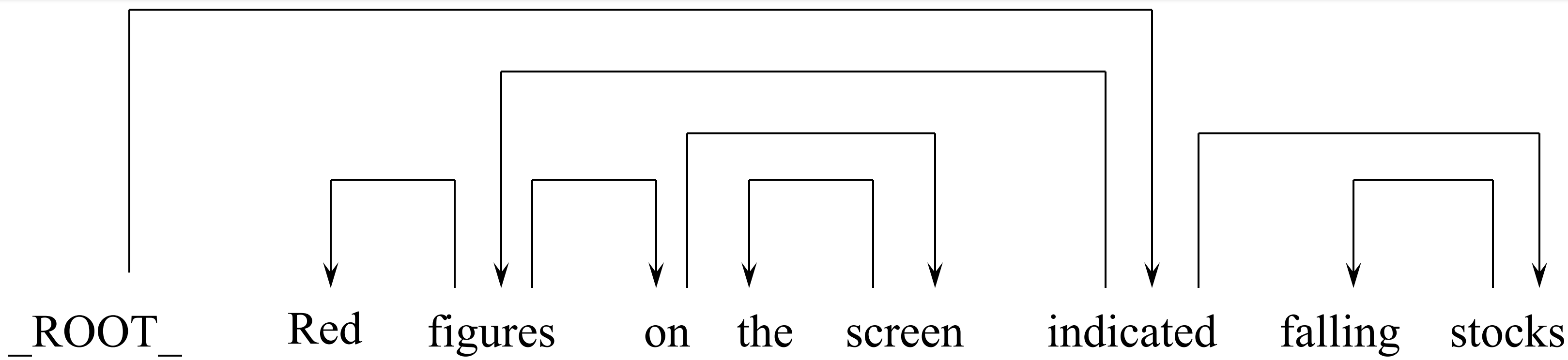


Example

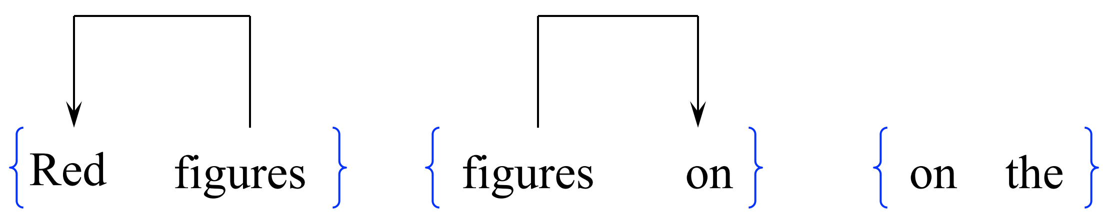


Spans:

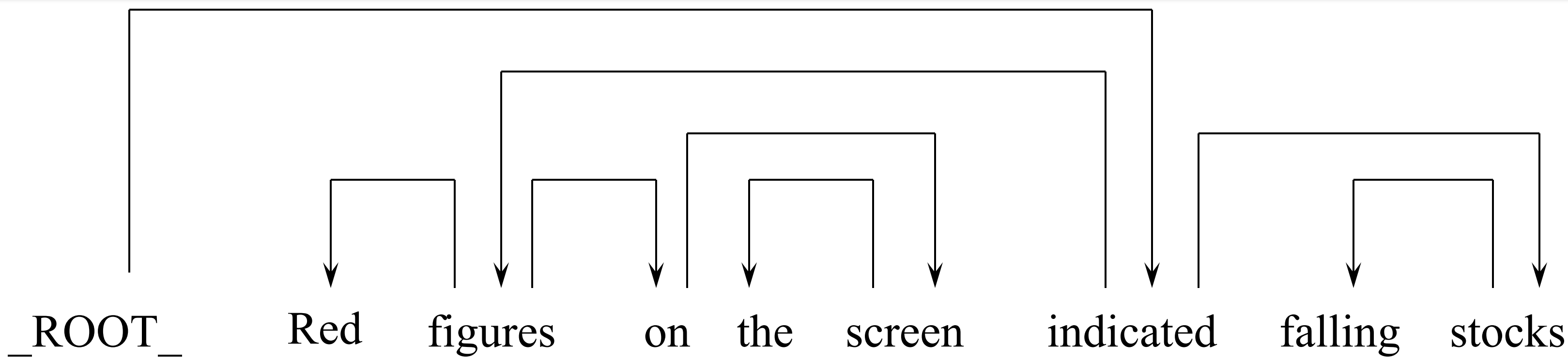
Assembly of correct parse



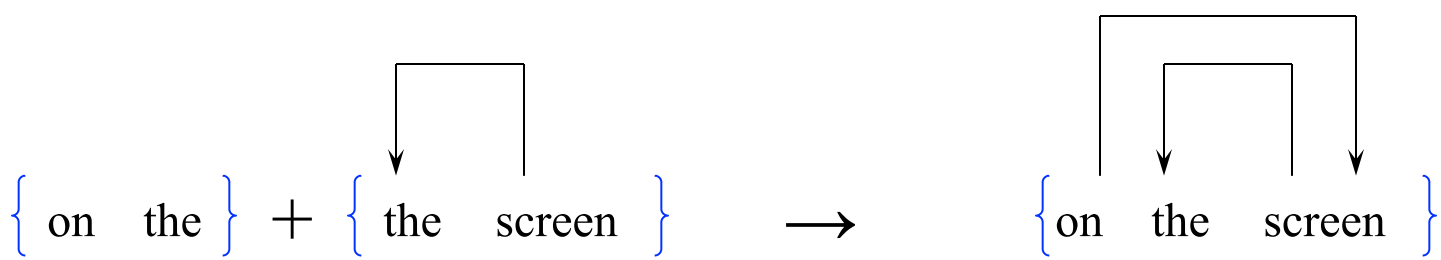
Start by combining adjacent words to minimal spans



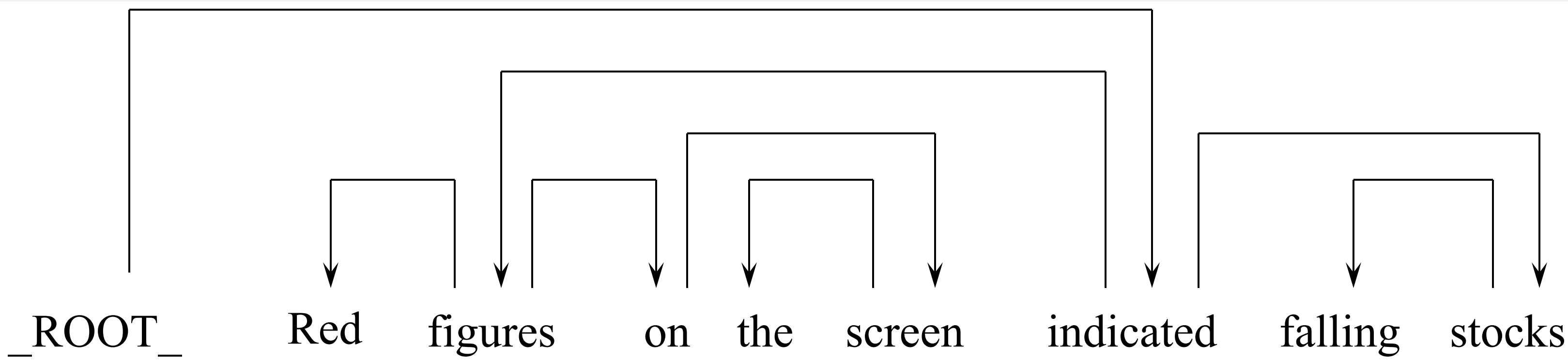
Assembly of correct parse



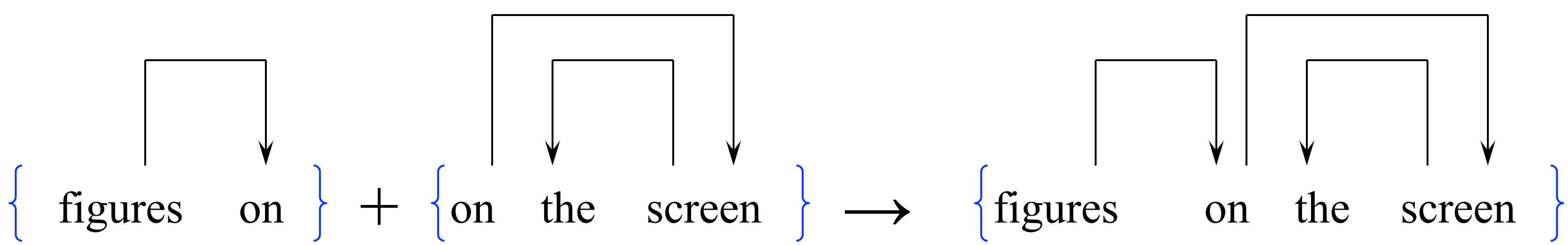
Combine spans which overlap in one word; this word must be governed by a word in the left or right span.



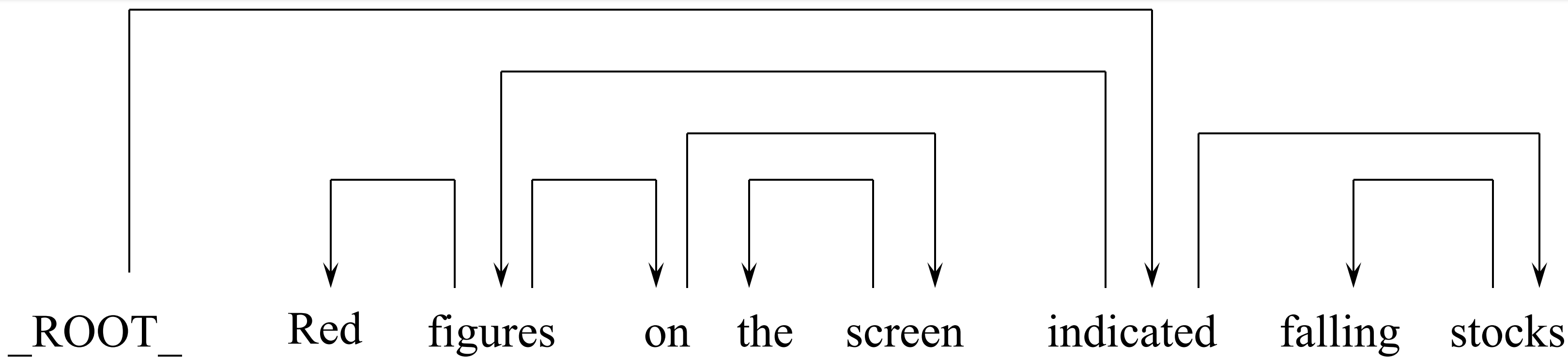
Assembly of correct parse



Combine spans which overlap in one word; this word must be governed by a word in the left or right span.

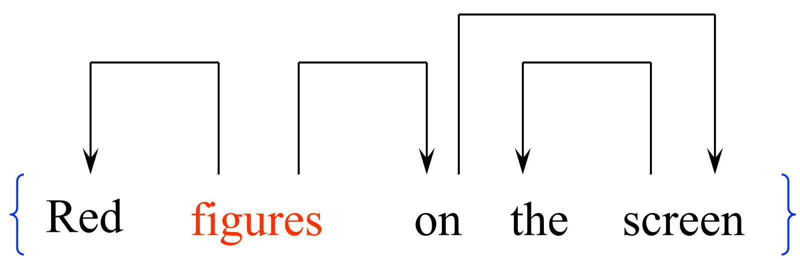


Assembly of correct parse

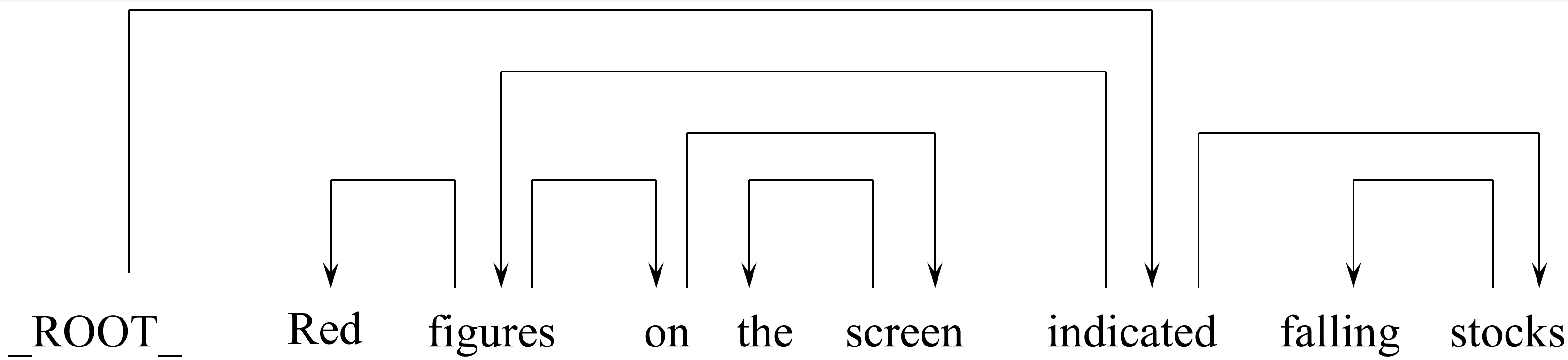


Combine spans which overlap in one word; this word must be governed by a word in the left or right span.

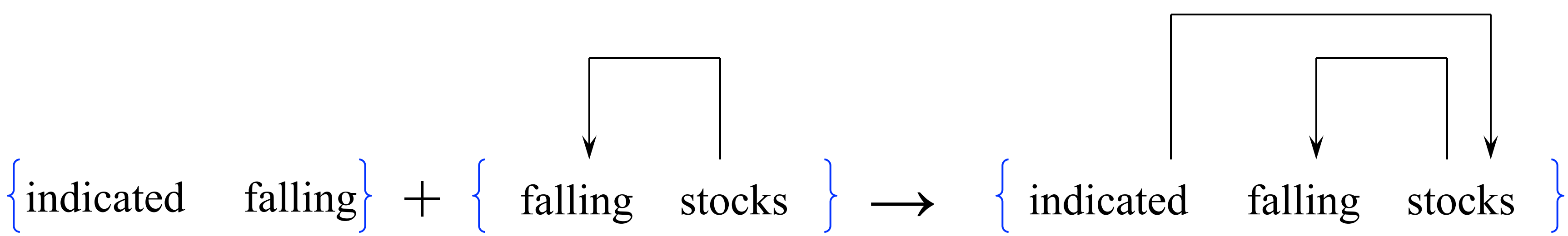
Invalid span



Assembly of correct parse



Combine spans which overlap in one word; this word must be governed by a word in the left or right span.



Features and Unification

Capturing Grammatical Features

A Simple Context Free Grammar Fragment

NP → Det N
NP → PropN

Det → a, the, this, those

N → book, dog, books, dogs
PropN → John, Mary

V → sneezed, visited, gave
eat, eats

S → NP VP

VP → V
(John sneezed)

VP → V NP
(John visited Mary)

VP → V NP NP
(John gave Mary a book)

VP → V NP PP
(John gave a book to Mary)

Agreement

Determiner/Noun Agreement

- This dog
- Those dogs

Our grammar also generates

- *This dogs
- *Those dog

Subject/Verb Agreement

- This dog eats
- Those dogs eat

Our grammar also generates

- *This dog eat
- *Those dogs eats

Encoding Number Agreement in CFGs

$NP_{sing} \rightarrow Det_{sing}\ N_{sing}$
$NP_{pl} \rightarrow Det_{pl}\ N_{pl}$

$VP_{pl} \rightarrow V_{pl}\ NP_{sing}$
$VP_{pl} \rightarrow V_{pl}\ NP_{pl}$
$VP_{sing} \rightarrow V_{sing}\ NP_{sing}$
$VP_{sing} \rightarrow V_{sing}\ NP_{pl}$

$S_{sing} \rightarrow NP_{sing}\ VP_{sing}$
$S_{pl} \rightarrow NP_{pl}\ VP_{pl}$

$Det_{sing} \rightarrow$ this
$Det_{pl} \rightarrow$ those

$N_{sing} \rightarrow$ dog
$N_{pl} \rightarrow$ dogs

$V_{sing} \rightarrow$ eats
$V_{pl} \rightarrow$ eat

Subcategorization

- Sneeze: John sneezed

  *John sneezed $[\text{the book}]_{NP}$
- Find: Please find $[\text{a flight to NY}]_{NP}$

  *Please find
- Give: Give $[\text{me}]_{NP}[\text{a cheaper fare}]_{NP}$

  *Give $[\text{with a flight}]_{PP}$
- Prefer: I prefer $[\text{to leave earlier}]_{TO\text{-}VP}$

  *I prefer $[\text{United has a flight}]_{S}$
- …

Possible CFG Solution

REPLACE:

- VP → V
- VP → V NP
- VP → V NP PP
- …

WITH:

- VP → $V_{Intrans}$
- VP → V_{Trans} NP
- VP → $V_{Trans+PP}$ NP PP
- $V_{Intrans}$ → sneeze
- V_{Trans} → find
- $V_{Trans+PP}$ → give

Encoding Number Agreement + Subcats…

- VP → $V_{Intrans/sing}$
- VP → $V_{Intrans/pl}$

- VP → $V_{Trans/sing}$ NP
- VP → $V_{Trans/pl}$ NP

- VP → $V_{Trans+PP/sing}$ NP PP
- VP → $V_{Trans+PP/pl}$ NP PP

- $V_{Intrans/sing}$ → sneezes
- $V_{Intrans/pl}$ → sneeze

- $V_{Trans/sing}$ → finds
- $V_{Trans/pl}$ → find

- $V_{Trans+PP/sing}$ → gives
- $V_{Trans+PP/pl}$ → give

But what about "I sneeze", "you sneeze", "he sneezes"….

Features, informally

View both words and grammar non-terminals as *complex objects*, each of which has a set of *associated property-value pairs* (called *features*) that can be manipulated.

- Det [num = sg] → this
- Det [num = pl] → those
- N [num = sg] → dog
- N [num =pl] → dogs

Then a grammar can contain:

NP → Det N *but only if Det [num] = N [num]*

Feature Agreement

OK:

NP → Det N *but only if Det [num] = N [num]*

Better:

NP → Det [num = α] N [num = α]

Best:

NP [num = α] → Det [num = α] N [num = α]

as well as

S → NP [num = α] VP [num = α]

Features and *Feature Structures*

- We can encode these properties by associating what are called *feature structures* with grammatical constituents.
- A feature structure is a set of *feature-value pairs* where:
  - *features* are atomic symbols
  - *values* are either atomic symbols or (recursively embedded) feature structures

$$\begin{bmatrix} Feature_1 & Value_1 \\ Feature_2 & Value_2 \\ \vdots & \vdots \\ Feature_n & Value_n \end{bmatrix}$$

Example Feature Structures

$$\begin{bmatrix} \textit{Number} & \textit{SG} \end{bmatrix}$$

$$\begin{bmatrix} \textit{Number} & \textit{SG} \\ \textit{Person} & \textit{3} \end{bmatrix}$$

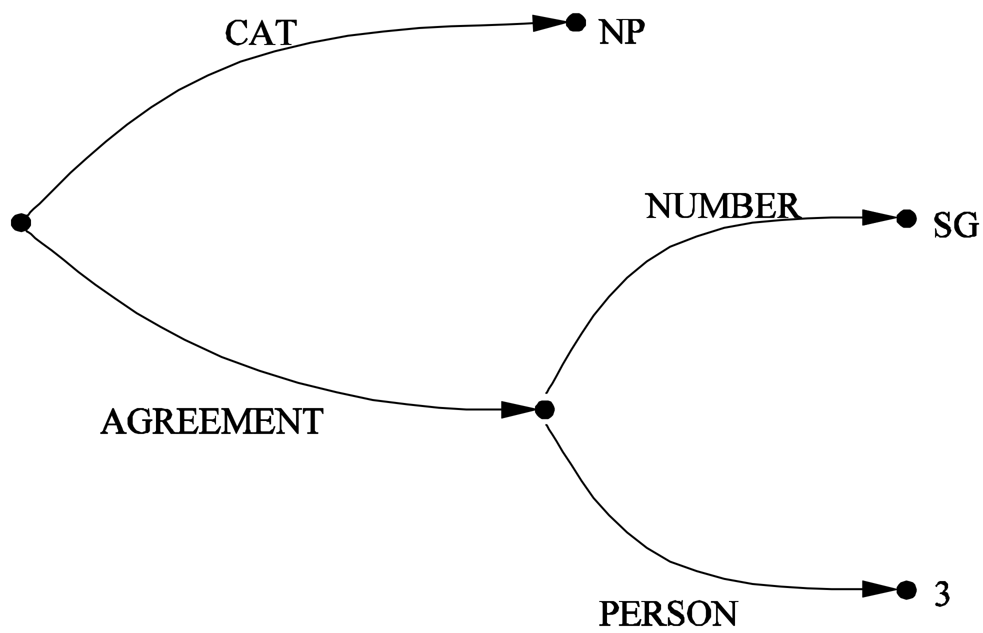
$$\begin{bmatrix} \textit{Cat} & \textit{NP} \\ \textit{Number} & \textit{SG} \\ \textit{Person} & \textit{3} \end{bmatrix}$$

Bundles of Features

- Feature Values can be feature structures themselves.
- This is useful when certain features commonly co-occur, as number and person.

$$\begin{bmatrix} \textit{Cat} & \textit{NP} \\ \textit{Agreement} & \begin{bmatrix} \textit{Number} & \textit{SG} \\ \textit{Person} & \textit{3} \end{bmatrix} \end{bmatrix}$$

Feature Structures as DAGs



Reentrant Structure

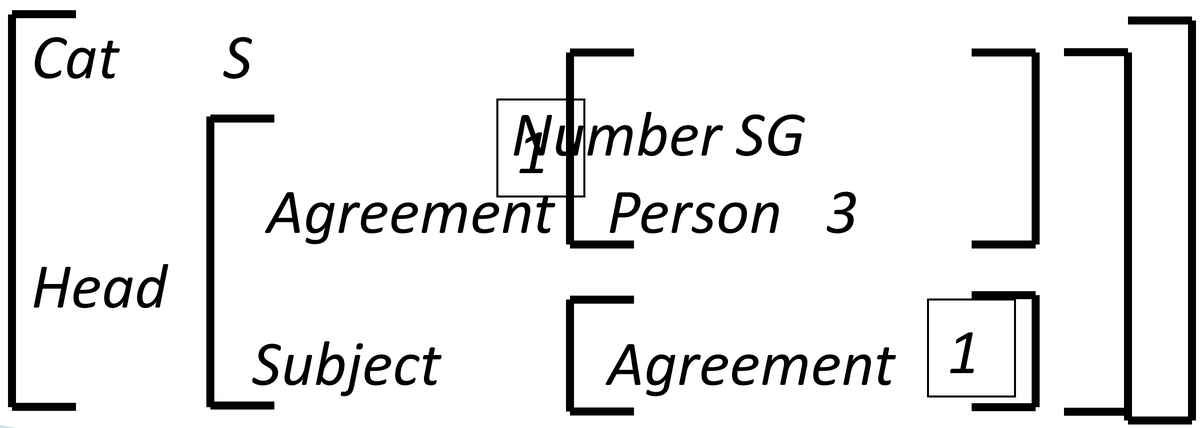
- Multiple features in a feature structure can *share* the same value. In this case they share structure, not just have the same value.

$$\begin{bmatrix} \textit{Cat} & \textit{S} \\ \textit{Head} & \begin{bmatrix} \textit{Agreement} & \boxed{1}\begin{bmatrix}\textit{Number} & \textit{SG}\\ \textit{Person} & \textit{3}\end{bmatrix} \\ \textit{Subject} & \begin{bmatrix}\textit{Agreement} & \boxed{1}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- Numerical indices indicate the shared value.

Feature Paths

- It will also be useful to talk about paths through feature structures. As in the paths
- <HEAD AGREEMENT NUMBER>
- <HEAD SUBJECT AGREEMENT NUMBER>



Unification I

Key operations on feature structures

1. check the compatibility of two structures
2. merge the information in two structures

We can do both with a single operation called *Unification*.

Unifying two feature structures produces a new feature structure that is more specific (has more information) than, or is identical to, each of the input feature structures.

The Unification Operation: *U*

- Two feature structures can be unified if the component features that make them up are *compatible*.

[number sg] U [number sg] = [number sg]
[number sg] U [number pl] = fails!

- Structures are *compatible* if they contain no features that are incompatible.
- If so, unification returns the union of all feature/ value pairs.

The Unification Operation

$[Number\ sg] \cup [Number\ [\ ]\ ] =$ **[***Number sg***]**

$[Number\ sg] \cup [Person\ 3] = \begin{bmatrix} Number & sg \\ Person & 3 \end{bmatrix}$

The Unification Operation

$$\begin{bmatrix} Agreement [Number\ sg] \\ Subject\quad [Agreement\quad [Number\ sg]\ ] \end{bmatrix}$$

U

$$[Subject\quad [Agreement\quad [Person\ 3]\ ]]$$

=

$$\begin{bmatrix} Agreement [Number\ sg] \\ Subject \begin{bmatrix} Agreement \begin{bmatrix} Number & sg \\ Person & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The Unification Operation

[Head [Subject [Agreement [Number pl]]]]

U

$$\begin{bmatrix} \textit{Cat} & \textit{S} \\ \textit{Head} & \begin{bmatrix} \textit{Agreement} & \boxed{1} \begin{bmatrix} \textit{Number sg} \\ \textit{Person} \;\; 3 \end{bmatrix} \\ \textit{Subject} & \textit{Agreement} \begin{bmatrix} \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

= Fail!

Properties of Unification

- *Monotonic*: if some description is true of a feature structure, it will still be true after unifying it with another feature structure.

- *Order independent (commutative)*: Unifying a set of feature structures in any order yields the same result.

Features, Unification, and Grammars

To incorporate all this into grammars:

- Each constituent has a feature-structure associated with it
- Each grammar rule has a (potentially empty) set of *unification constraints* associated with it.
  - The set of unification constraints must be satisfied for the rule to be satisfied.

Unification Constraints

$X_0 \rightarrow X_1 \ldots X_n$ } *Grammar rule*

$< X_i$ feature path $>$
= atomic value
$< X_i$ feature path $>$
= $< X_k$ feature path $>$

} ***Set of constraints***

Agreement

NP → Det Nominal
< Det AGREEMENT > = < Nominal AGREEMENT >
< NP AGREEMENT > = < Nominal AGREEMENT >

Noun → flight
< Noun AGREEMENT NUMBER > = SG

Noun → flights
< Noun AGREEMENT NUMBER > = PL

Nominal → Noun
< Nominal AGREEMENT > = < Noun AGREEMENT >

Det → this
< Det AGREEMENT NUMBER > = SG

Unification and Parsing

- Assume we've augmented our grammar with sets of unification constraints.

- What changes do we need to make to a parser to make use of them?
  1. *Build feature structures* and associate each with a subtree
  2. *Unify feature structures* as subtrees are created from smaller subtrees
  3. *Block ill-formed constituents*

Unification and Earley Parsing

With respect to an Earley-style parser…

- Build feature structures (represented as DAGs) and associate them with states in the chart
- Unify feature structures as states are advanced in the chart
- Block ill-formed states from entering the chart

Building Feature Structures

- Features of most grammatical categories are copied from head child to parent
  - (e.g., from V to VP, Nom to NP, N to Nom)

VP → V NP

< VP HEAD > = < V HEAD >

S → NP VP

< NP HEAD AGREEMENT > = < VP HEAD AGREEMENT>

< S HEAD > = < VP HEAD >

$$\begin{bmatrix} S & [head\ \boxed{1}\ ] \\ NP & [head\ [agreement\ \boxed{2}\ ]] \\ VP & [head\ \boxed{1}\ [agreement\ \boxed{2}\ ]] \end{bmatrix}$$

Augmenting States with DAGs

- We just add a new field to the representation of the states

S → . NP VP, [0,0], Dag

Example

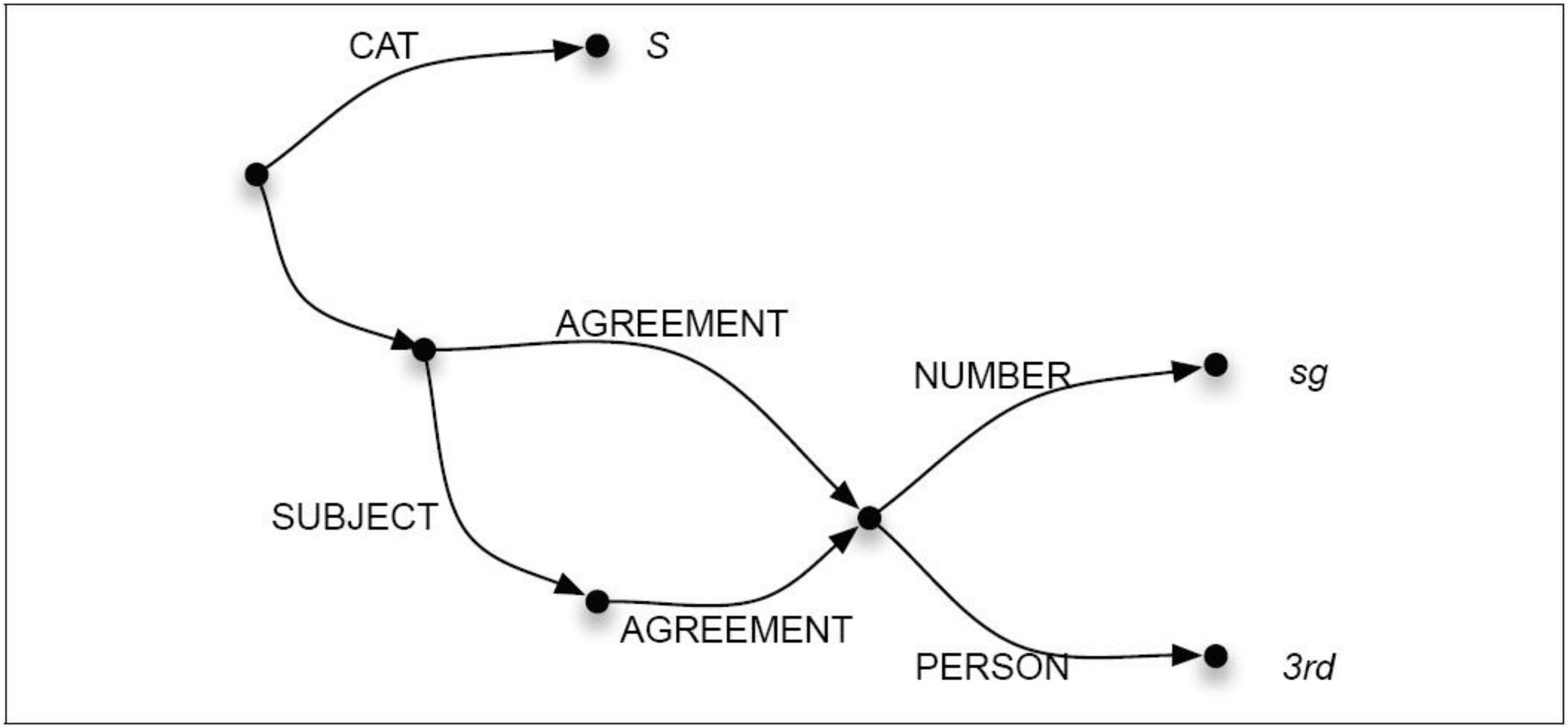
- NP → Det . Nominal [0,1], DAG1

$$\begin{bmatrix} np & [head \; \boxed{1} \;] \\ det & [head \; [agreement \; \boxed{2} \; [number \; sg]]] \\ Nominal & [head \; \boxed{1} \; [agreement \; \boxed{2} \;]] \end{bmatrix}$$

Nominal → Noun ., [1,2], DAG2

$$\begin{bmatrix} Nominal & [head \; \boxed{1} \;] \\ noun & [head \; \boxed{1} \; [agreement \; [number \; sg]]] \end{bmatrix}$$

Figure 15.2



Unifying States and Blocking

- Keep much of the Earley Algorithm the same.
- We want to unify the DAGs of existing states as they are combined as specified by the grammatical constraints.

- Alter COMPLETER – when a new state is created, first make sure the individual DAGs unify. If so, then add the new DAG (resulting from the unification) to the new state.

Unification for Semantics

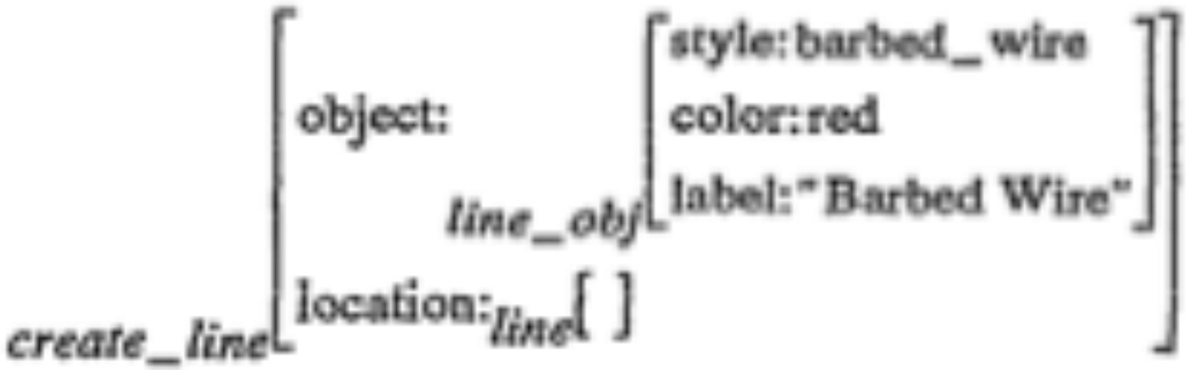


Figure 9: Feature Structure for 'barbed wire'

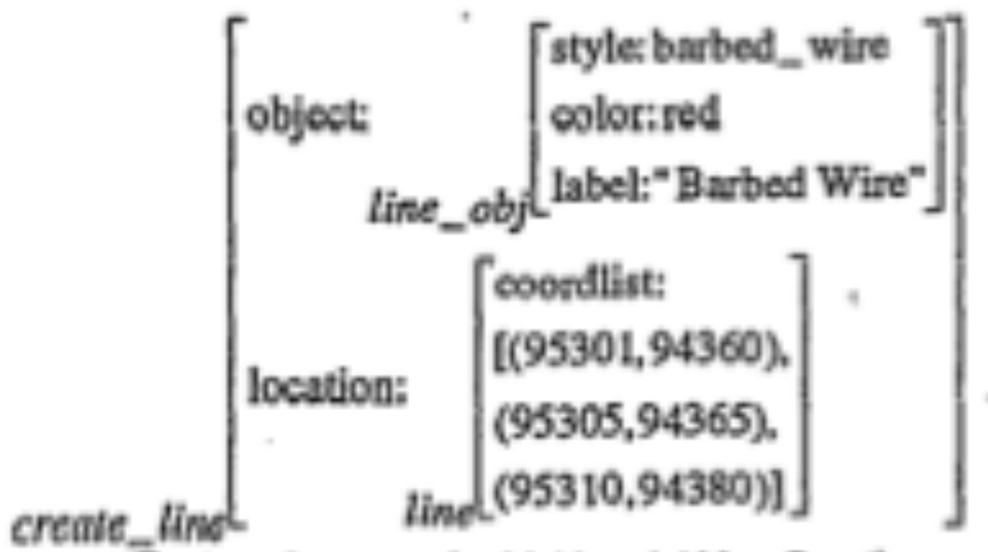


Figure 12: Feature Structure for Multimodal Line Creation