Feature Structures

Problems with CFGs

- We know that CFGs cannot handle certain things which are available in natural languages.
- In particular, CFGs cannot handle very well:
  - agreement
  - subcategorization
- We will look at a constraint-based representation schema which will allow us to represent fine-grained information such as:
  - number/person agreement
  - subcategorization
  - semantic categories like mass/count

Agreement Problem

- What is the problem with the following CFG rules:

  S → NP VP

  NP → Det NOMINAL

  NP → Pronoun

- *Answer*: Since these rules do not enforce number and person agreement constraints, they over-generate and allow the following constructs:

  * They sleeps

  * He sleep

  * A dogs

  * These dog

An Awkward Solution to Agreement Problem

- One way to handle the agreement phenomena in a strictly context-free approach is to encode the constraints into the non-terminal categories and then into CFG rules.
- For example, our grammar will be:

  S → SgS | PlS

  SgS → SgNP SgVP

  PlS → PlNP PlVP

  SgNP → SgDet SgNOMINAL

  SgNP → SgPronoun

  PlNP → PlDet PlNOMINAL

  PlNP → PlPronoun

- This solution will explode the number of non-terminals and rules. The resulting grammar will not be a clean grammar.

Subcategorization Problem

- What is the problem with the following CFG rules:

  VP → Verb

  VP → Verb NP

- *Answer*: Since these rules do not enforce subcategorization constraints, they over-generate and allow the following constructs:

  * They take

  * They sleep a glass

An Awkward Solution to Subcategorization Problem

- Again, one way to handle the subcategorization phenomena in a strictly context-free approach is to encode the constraints into the non-terminal categories and then into CFG rules.
- For example, our grammar will be:

  VP → IntransVP | TransVP

  IntransVP → IntransVerb

  TransVP → TransVerb NP

- This solution will again explode the number of non-terminals and rules.
- Remember that we may almost 100 subcategorizations for English verbs. The resulting grammar will not be a clean grammar.

A Better Solution

- A better solution for agreement and subcategorization problems is to treat terminals and non-terminals as complex objects with associated properties (called **features**) that can be manipulated.
- So, we may code rules as follows: (not CF rules anymore)

S $\rightarrow$ NP VP *Only if the number of the NP is equal to the number of the VP.*

- Where *number of* are **features** of NP and VP, and they are manipulated (they are checked to see whether they are equal or not) by the rule above.

Feature Structures

- We can encode the properties associated with grammatical constituents (terminals and non-terminals) by using **Feature Structures**.
- A **feature structure** is a set of **feature-value** pairs.
  - A **feature** is an atomic symbol.
  - A **value** is either an atomic value or another feature structure.
- A feature structure can be illustrated by a matrix-like diagram (called **attribute-value matrix**).

$$\begin{bmatrix} Feature-1 & Value-1 \\ Feature-2 & Value-2 \\ . & \\ Feature-n & Value-n \end{bmatrix}$$

Example - Feature Structures

$$\begin{bmatrix} NUMBER & SG \end{bmatrix}$$

$$\begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix}$$

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

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

Reentrant Feature Structures

- We will allow multiple features in a feature structure to share the same values.
- They share the same structures not just that they have same value.

$$\begin{bmatrix} CAT & S \\ HEAD & \begin{bmatrix} AGREEMENT & (1)\begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix} \\ SUBJECT & \begin{bmatrix} AGREEMENT & (1) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Feature Path

- A **feature path** is a list of features through a feature structure leading to a particular value.
- For example,

  <HEAD AGREEMENT NUMBER> leads to SG

  <HEAD SUBJECT AGREEMENT PERSON> leads to 3

- We will use feature paths in the constraints of the rules.

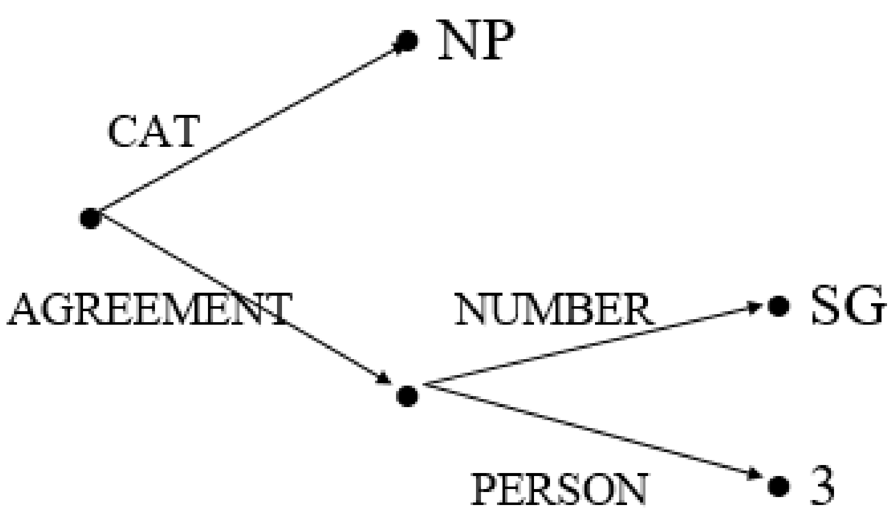
  S $\rightarrow$ NP VP

  <NP AGREEMENT> = <VP AGREEMENT>

DAG Representation of Feature Structures

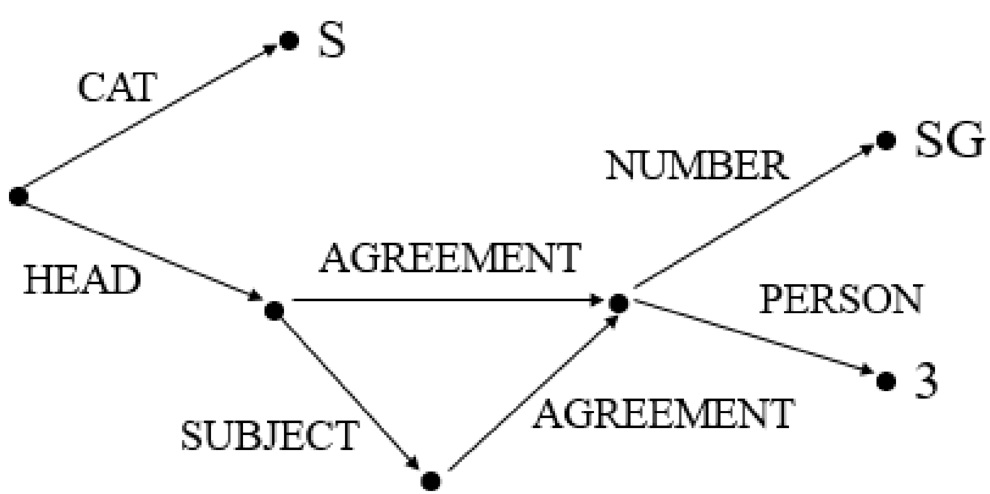
- A feature structure can also be represented by using a DAG (directed acyclic graph).

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



DAG of A Reentrant Feature Structure

$$\begin{bmatrix} CAT & S \\ HEAD & \begin{bmatrix} AGREEMENT & (1)\begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix} \\ SUBJECT & \begin{bmatrix} AGREEMENT & (1) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$



Unification of Feature Structures

- By the unification of feature structures, we will:
  - Check the compatibility of two feature structures.
  - Merge the information in two feature structures.
- The result of a unification operation of two feature structures can be:
  - unifiable -- they will merge into a single feature structure
  - fails -- if two feature structures are not compatible.
- We will look at how does this unification process perform the above tasks.

Unification Example

- We say that two feature structures can be unified if two feature structures that make them up are compatible.

$$[NUMBER \quad SG] \cup [NUMBER \quad SG] = [NUMBER \quad SG]$$

succeeds

$$[NUMBER \quad SG] \cup [NUMBER \quad PL]$$

fails

Unification Operator

Unification Example (cont.)

- The unification process can bind an undefined value to a value, or can merge the information in two feature structures.

$$\begin{bmatrix} NUMBER & SG \end{bmatrix} \cup \begin{bmatrix} NUMBER & [] \end{bmatrix} = \begin{bmatrix} NUMBER & SG \end{bmatrix}$$

$$\begin{bmatrix} NUMBER & SG \end{bmatrix} \cup \begin{bmatrix} PERSON & 3 \end{bmatrix} = \begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix}$$

Unification Example -- Complex Structures

$$\begin{bmatrix} AGREEMENT & (1) \\ SUBJECT & \begin{bmatrix} AGREEMENT & (1) \end{bmatrix} \end{bmatrix} \cup$$

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

$$= \begin{bmatrix} AGREEMENT & (1) \\ SUBJECT & \begin{bmatrix} AGREEMENT & (1) \begin{bmatrix} PERSON & 3 \\ NUMBER & SG \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Subsumption

- A more abstract (less specific) feature structure **subsumes** an equally or more specific one.
- **Subsumption** is represented by the operator $\sqsubseteq$
- A feature structure F **subsumes** a feature structure G (F $\sqsubseteq$ G) if and only if :
  - For every structure x in F, F(x) $\sqsubseteq$ G(x) (where F(x) means the value of the feature x of the feature structure F).
  - For all paths p and q in F such that F(p)=F(q), it is also the case that G(p)=G(q).

Subsumption Example

Consider the following feature structures:

(1) $\begin{bmatrix} NUMBER & SG \end{bmatrix}$

(2) $\begin{bmatrix} PERSON & 3 \end{bmatrix}$

(3) $\begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix}$

(1) $\subseteq$ (3)

(2) $\subseteq$ (3)

but there is no subsumption relation between (1) and (2)

Feature Structures in The Grammar

- We will incorporate the feature structures and the unification process as follows:
  - All constituents (non-terminals) will be associated with feature structures.
  - Sets of unification constraints will be associated with grammar rules, and these rules must be satisfied for the rule to be satisfied.
- These attachments accomplish the following goals:
  - To associate feature structures with both lexical items and instances of grammatical categories.
  - To guide the composition of feature structures for larger grammatical constituents based on the feature structures of their component parts.
  - To enforce compatibility constraints between specified parts of grammatical constraints

Unification Constraints

- Each grammar rule will be associated with a set of unification constraints.

$\beta_0 \rightarrow \beta_1 \ldots \beta_n$ {set of unification constraints}

- Each unification constraint will be in one of the following forms.

< β_i feature path> = Atomic value

< β_i feature path> = < β_j feature path>

Unification Constraints -- Example

- For example, the following rule

S $\rightarrow$ NP VP

Only if the number of the NP is equal to the number of the VP.

will be represented as follows:

S $\rightarrow$ NP VP

<NP NUMBER> = <VP NUMBER>

Agreement Constraints

S $\rightarrow$ NP VP

<NP NUMBER> = <VP NUMBER>

S $\rightarrow$ Aux NP VP

<Aux AGREEMENT> = <NP AGREEMENT>

NP $\rightarrow$ Det NOMINAL

<Det AGREEMENT> = <NOMINAL AGREEMENT>

<NP AGREEMENT> = <NOMINAL AGREEMENT>

NOMINAL $\rightarrow$ Noun

<NOMINAL AGREEMENT> = <Noun AGREEMENT>

VP $\rightarrow$ Verb NP

<VP AGREEMENT> = <Verb AGREEMENT>

Agreement Constraints -- Lexicon Entries

Aux $\rightarrow$ does	<Aux AGREEMENT NUMBER> = SG
	<Aux AGREEMENT PERSON> = 3
Aux $\rightarrow$ do	<Aux AGREEMENT NUMBER> = PL
Det $\rightarrow$ these	<Det AGREEMENT NUMBER> = PL
Det $\rightarrow$ this	<Det AGREEMENT NUMBER> = SG
Verb $\rightarrow$ serves	<Verb AGREEMENT NUMBER> = SG
	<Verb AGREEMENT PERSON> = 3
Verb $\rightarrow$ serve	<Verb AGREEMENT NUMBER> = PL
Noun $\rightarrow$ flights	<Noun AGREEMENT NUMBER> = PL
Noun $\rightarrow$ flight	<Noun AGREEMENT NUMBER> = SG

Head Features

- Certain features are copied from children to parent in feature structures.
- For example, AGREEMENT feature in NOMINAL is copied into NP.
- The features for most grammatical categories are copied from one of the children to the parent.
- The child that provides the features is called **head of the phrase**, and the features copied are referred to as **head features**.
- A verb is a head of a verb phrase, and a nominal is a head of a noun phrase. We may reflect these constructs in feature structures as follows:

NP $\rightarrow$ Det NOMINAL

<Det HEAD AGREEMENT> = <NOMINAL HEAD AGREEMENT>

<NP HEAD> = <NOMINAL HEAD>

VP $\rightarrow$ Verb NP

<VP HEAD> = <Verb HEAD>

SubCategorization Constraints

- For verb phrases, we can represent subcategorization constraints using three techniques:
  - Atomic Subcat Symbols
  - Encoding Subcat lists as feature structures
  - Minimal Rule Approach (using lists directly)
- We may use any of these representations.

Atomic Subcat Symbols

VP $\rightarrow$ Verb

<VP HEAD> = <Verb HEAD>

<VP HEAD SUBCAT> = INTRANS

VP $\rightarrow$ Verb NP

<VP HEAD> = <Verb HEAD>

<VP HEAD SUBCAT> = TRANS

VP $\rightarrow$ Verb NP NP

<VP HEAD> = <Verb HEAD>

<VP HEAD SUBCAT> = DITRANS

Verb $\rightarrow$ slept <Verb HEAD SUBCAT> = INTRANS

Verb $\rightarrow$ served <Verb HEAD SUBCAT> = TRANS

Verb $\rightarrow$ gave <Verb HEAD SUBCAT> = DITRANS

Encoding Subcat Lists as Features

Verb → gave

<Verb HEAD SUBCAT FIRST CAT> = NP

<Verb HEAD SUBCAT SECOND CAT> = NP

<Verb HEAD SUBCAT THIRD> = END

VP → Verb NP NP

<VP HEAD> = <Verb HEAD>

<VP HEAD SUBCAT FIRST CAT> = <NP CAT>

<VP HEAD SUBCAT SECOND CAT> = <NP CAT>

<VP HEAD SUBCAT THIRD> = END

- We are only encoding lists using positional features

Minimal Rule Approach

- In fact, we do not use symbols like SECOND, THIRD. They are just used to encode lists. We can use lists directly (similar to LISP).

<SUBCAT FIRST CAT> = NP

<SUBCAT REST FIRST CAT> = NP

<SUBCAT REST REST> = END

Subcategorization Frames for Lexical Entries

- We can use two different notations to represent subcategorization frames for lexical entries (verbs).

Verb $\rightarrow$ want

<Verb HEAD SUBCAT FIRST CAT> = NP

Verb $\rightarrow$ want

<Verb HEAD SUBCAT FIRST CAT> = VP

<Verb HEAD SUBCAT FIRST FORM> = INFINITITIVE

$$\begin{bmatrix} ORTH & WANT \\ CAT & VERB \\ HEAD & \begin{bmatrix} SUBCAT & < \begin{bmatrix} CAT & NP \end{bmatrix}, \begin{bmatrix} CAT & VP \\ HEAD & \begin{bmatrix} VFORM & INFINITIVE \end{bmatrix} \end{bmatrix} > \end{bmatrix} \end{bmatrix}$$

Implementing Unification

- The representation we have used cannot facilitate the destructive merger aspect of unification algorithm.
- For this reason, we add additional features (additional edges to DAGs) into our feature structures.
- Each feature structure will consists of two fields:
  - **Content** Field -- This field can be NULL or may contain ordinary feature structure.
  - **Pointer** Field -- This field can be NULL or may contain a pointer into another feature structure.
- If the pointer field of a DAG is NULL, the content field of DAG contains the actual feature structure to be processed.
- If the pointer field of a DAG is not NULL, the destination of that pointer represents the actual feature structure to be processed.

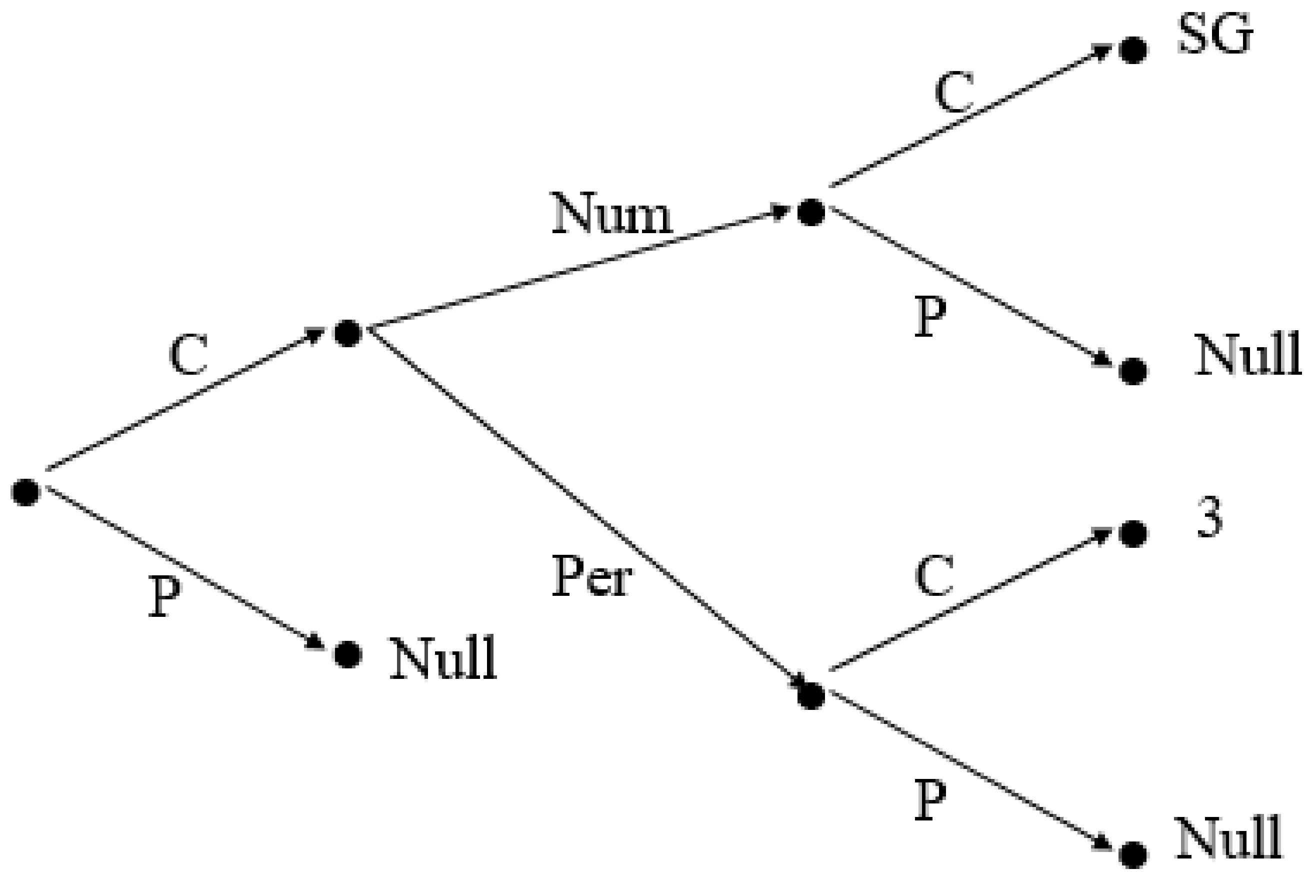
Extended Feature Structures

$$\begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix}$$

$$\Downarrow$$

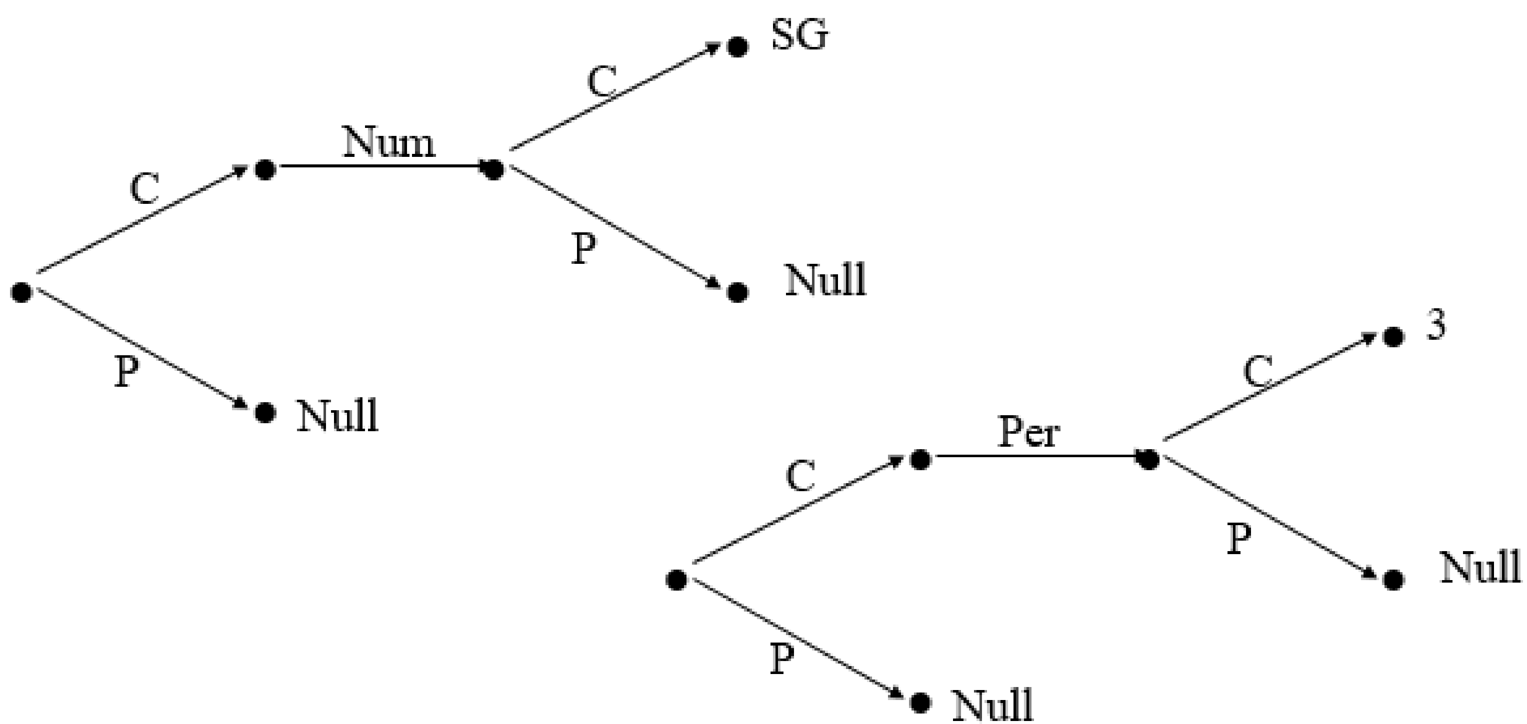
$$\begin{bmatrix} CONTENT & \begin{bmatrix} NUMBER & \begin{bmatrix} CONTENT & SG \\ POINTER & NULL \end{bmatrix} \\ PERSON & \begin{bmatrix} CONTENT & 3 \\ POINTER & NULL \end{bmatrix} \end{bmatrix} \\ POINTER & NULL \end{bmatrix}$$

Extended DAG

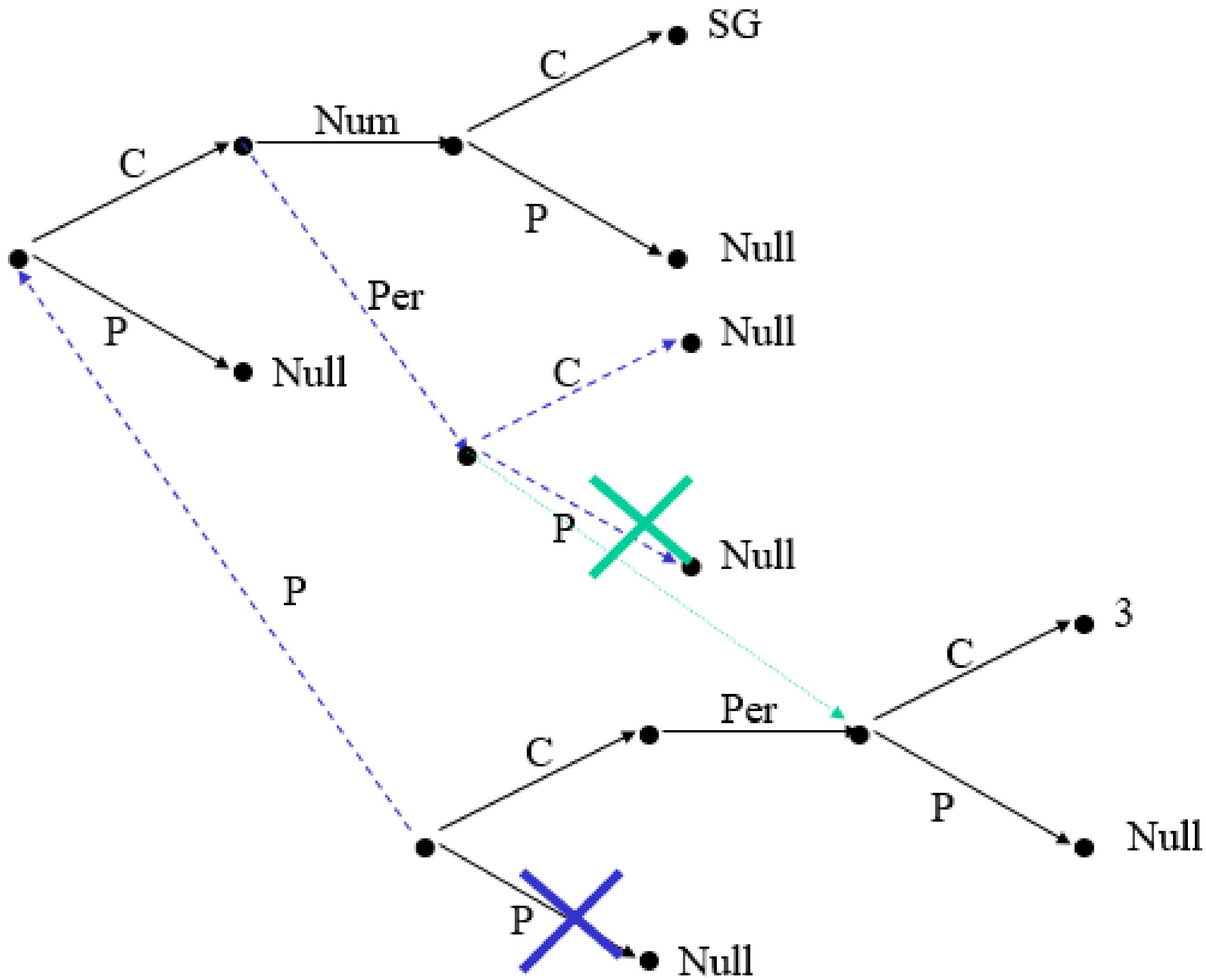


Unification of Extended DAGs

$$\begin{bmatrix} NUMBER & SG \end{bmatrix} \cup \begin{bmatrix} PERSON & 3 \end{bmatrix} = \begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix}$$



Unification of Extended DAGs (cont.)



Unification Algorithm

```
function  UNIFY(f1,f2)  returns  fstructure or failure
f1real ← real contents of f1  /* dereference f1 */
f2real ← real contents of f2  /* dereference f2 */
if  f1real is Null then  { f1.pointer ←  f2;  return f2; }
else if  f2real is Null  then  { f2.pointer ←  f1;  return f1; }
else if  f1real and f2real are identical  then  { f1.pointer ←  f2;  return f2; }
else if  f1real and f2real are complex feature structures  then
    { f2.pointer ←  f1;
      for each  feature  in  f2real  do
          {  otherfeature ← Find or create a feature corresponding to feature in f1real;
             if  UNIFY(feature.value,otherfeature.value)  returns  failure then
                 return failure;  }
      return  f1;  }
else return failure;
```

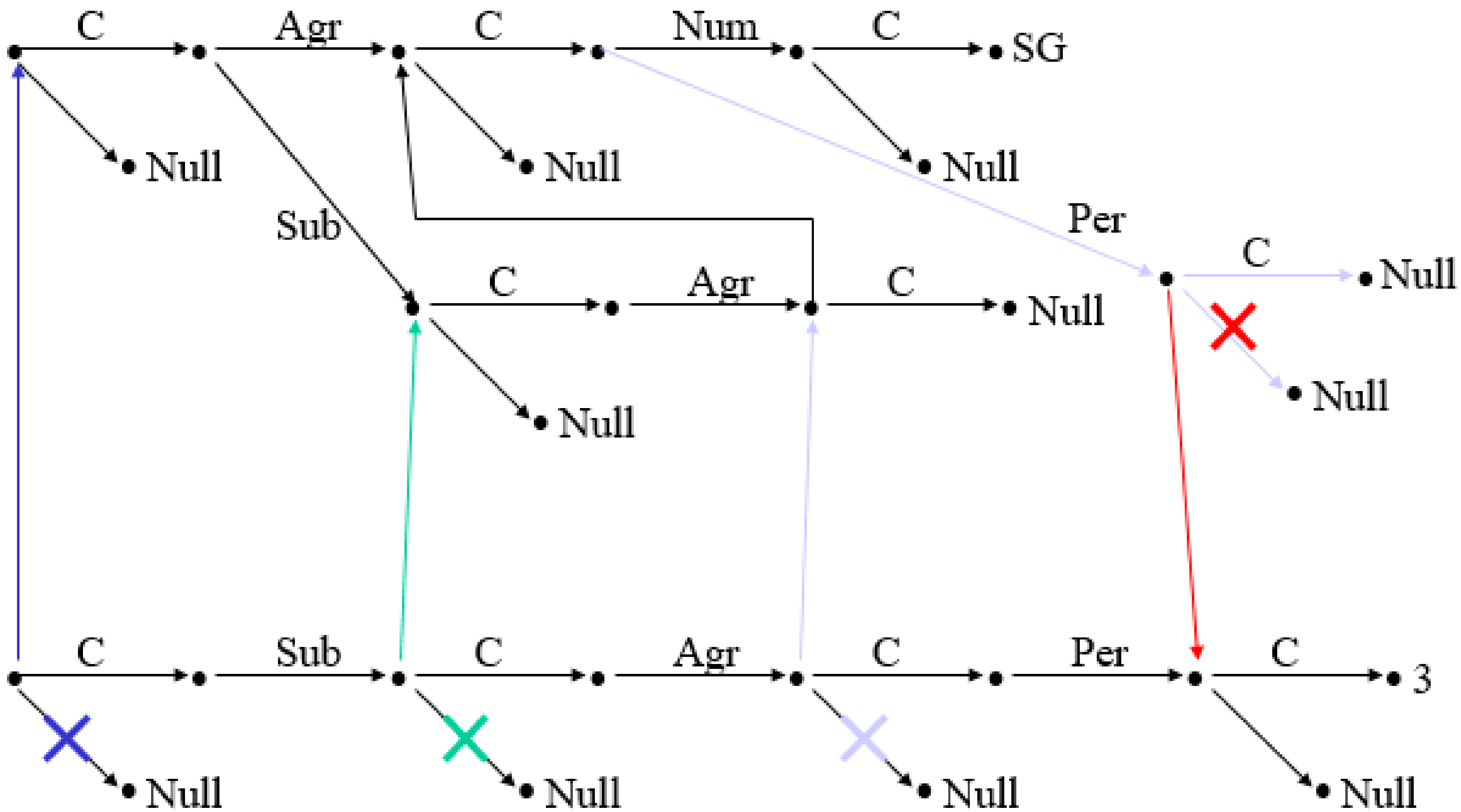
Example - Unification of Complex Structures

$$\begin{bmatrix} AGREEMENT & (1)\begin{bmatrix} NUMBER & SG \end{bmatrix} \\ SUBJECT & \begin{bmatrix} AGREEMENT & (1) \end{bmatrix} \end{bmatrix} \cup$$

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

$$= \begin{bmatrix} AGREEMENT & (1)\begin{bmatrix} NUMBER & SG \\ PERSON & 3 \end{bmatrix} \\ SUBJECT & \begin{bmatrix} AGREEMENT & (1) \end{bmatrix} \end{bmatrix}$$

Example - Unification of Complex Structures (cont.)



Parsing with Unification Constraints

- Let us assume that we have augmented our grammar with sets of unification constraints.

- What changes do we need to make a parser to make use of them?
  - Building feature structures and associate them with sub-trees.
  - Unifying feature structures when sub-trees are created.
  - Blocking ill-formed constituents

Earley Parsing with Unification Con

- What do we have to do to integrate unification constraints with Earley Parser?
  - Building feature structures (represented as DAGs) and associate them with states in the chart.
  - Unifying feature structures as states are advanced in the chart.
  - Blocking ill-formed states from entering the chart.
- The main change will be in COMPLETER function of Earley Parser. This routine will invoke the unifier to unify two feature structures.

Building Feature Structures

NP $\rightarrow$ Det NOMINAL

<Det HEAD AGREEMENT> = <NOMINAL HEAD AGREEMENT>

<NP HEAD> = <NOMINAL HEAD>

corresponds to

$$\begin{bmatrix} NP & \begin{bmatrix} HEAD & (1) \end{bmatrix} \\ Det & \begin{bmatrix} HEAD & \begin{bmatrix} AGREEMENT & (2) \end{bmatrix} \end{bmatrix} \\ NOMINAL & \begin{bmatrix} HEAD & (1) \begin{bmatrix} AGREEMENT & (2) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Augmenting States with DAGs

- Each state will have an additional field to contain the DAG representing the feature structure corresponding to the state.
- When a rule is first used by PREDICTOR to create a state, the DAG associated with the state will simply consist of the DAG retrieved from the rule.
- For example,

$S \rightarrow \bullet$ NP VP, [0,0],[],Dag_1

where Dag_1 is the feature structure corresponding to S $\rightarrow$ NP VP.

NP $\rightarrow \bullet$ Det NOMINAL, [0,0],[],Dag_2

where Dag_2 is the feature structure corresponding to S $\rightarrow$ Det NOMINAL.

What does COMPLETER do?

- When COMPLETER advances the dot in a state, it should unify the feature structure of the newly completed state with the appropriate part of the feature structure being advanced.
- If this unification process is succesful, the new state gets the result of the unification as its DAG, and this new state is entered into the chart.
  - If it fails, nothing is entered into the chart.

A Completion Example

Parsing the phrase *that flight* after *that* is processed.

$NP \rightarrow Det \bullet NOMINAL, [0,1],[SDet],Dag_1$

$$Dag_1 \begin{bmatrix} NP & [HEAD \quad (1)] \\ Det & [HEAD \quad [AGREEMENT \quad (2)[NUMBER \quad SG]]] \\ NOMINAL & [HEAD \quad (1)[AGREEMENT \quad (2)]] \end{bmatrix}$$

A newly completed state

$NOMINAL \rightarrow Noun \bullet, [1,2],[SNoun],Dag_2$

$$Dag_2 \begin{bmatrix} NOMINAL & [HEAD \quad (1)] \\ Noun & [HEAD \quad (1)[AGREEMENT \quad [NUMBER \quad SG]]] \end{bmatrix}$$

To advance in NP, the parser unifies the feature structure found under the NOMINAL feature of Dag2, with the feature structure found under the NOMINAL feature of Dag1.

Earley Parse

```
function  EARLEY-PARSE(words,grammar)  returns  chart
    ENQUEUE((γ  → • S, [0,0], chart[0],dagγ)
    for  i  from  0  to LENGTH(words)  do
      for each  state  in  chart[i]  do
        if  INCOMPLETE?(state) and NEXT-CAT(state) is not a PS  then
            PREDICTOR(state)
        elseif INCOMPLETE?(state) and NEXT-CAT(state) is a PS  then
            SCANNER(state)
        else
            COMPLETER(state)
      end
    end
    return(chart)
```

Predictor and Scanner

procedure PREDICTOR(($A \rightarrow \alpha \bullet B\ \beta$, [i,j],dagA))

for each ($B \rightarrow \gamma$) **in** GRAMMAR-RULES-FOR(B,grammar) **do**

ENQUEUE(($B \rightarrow \bullet\ \gamma$, [i,j],dagB), chart[j])

end

procedure SCANNER(($A \rightarrow \alpha \bullet B\ \beta$, [i,j],dagA))

if ($B \in$ PARTS-OF-SPEECH(word[j]) **then**

ENQUEUE(($B \rightarrow$ word[j] $\bullet$, [j,j+1],dagB), chart[j+1])

end

Completer and UnifyStates

procedure COMPLETER(($B \rightarrow \gamma \bullet$, [j,k],dagB))

for each ($A \rightarrow \alpha \bullet B\ \beta$, [i,j],dagA) **in** chart[j] **do**

if newdag ← UNIFY-STATES(dagB,dagA,B) ≠ fails **then**
ENQUEUE(($A \rightarrow \alpha\ B \bullet \beta$, [i,k],newdag), chart[k])

end

procedure UNIFY-STATES(dag1,dag2,cat)

dag1cp ← CopyDag(dag1);

dag2cp ← CopyDag(dag2);

UNIFY(FollowPath(cat,dag1cp),FollowPath(cat,dag2cp));

end

Enqueue

procedure ENQUEUE(*state*,*chart-entry*)

if *state* is not subsumed by a state in *chart-entry* **then**

Add *state* at the end of *chart-entry*

end