Natural Language Processing II (SC674)

Prof. Feng Zhiwei

Ch4. Feature and Unification

4.1 Feature Structures in grammar

4.1.1 Attribute-value matrix

From a reductionist perspective, the history of the natural sciences over the last few hundred years can be seen as an attempt to explain the behavior of larger structures by the combined action of smaller primitives.

Biology: Cell action → Genes action → DNA action

Physics: Molecular → Atom → subatomic particles

It can be called as "**Reductionism**".

In NLP, we also are influenced by this reductionism.

E,g. In Chapter 2, we have proposed the following rule

S → Aux NP VP

It can be replaced by two rules of following form:

S → 3sgAux 3sgNP VP

S → Non-3sgAux Non3sgNP VP

Lexicon rules:

3sgAux → does | has | can | …

Non3sgAux → do | have | can |

We attempt to combine the smaller structures actions to explain the action of larger structures.

We shall use the **feature structures** to describe reductionism in NLP.

The feature structures are simply sets of feature value pairs, where features are un-analyzable atomic symbols drawn from some finite set, and values are either atomic symbols or feature structures.

The feature structures are illustrated with an **Attribute-Value Matrix** (AVM) as follows:

FEATURE1	VALUE1
FEATURE2	VALUE2
FEATUREn	VALUEn

E.g. 3sgNP can be illustrated by following AVM:

cat	NP
num	sig
person	3

3sgAux can be illustrated by following AVM:

cat	Aux
num	sing
per	3

In the feature structures, the features are not limited to atomic symbols as their values; they can also have other feature structures as their values.

It is very useful when we wish to bundle a set of feature-value pairs together for similar treatment. E,g, The feature “num” and “per” are often lumped together since grammatical subject must agree with their predicates in both of their number and person. This lumping together can introduce the feature “agreement” that takes a feature structure consisting of the number and person feature-value pairs as its value.

The feature structure of 3sgNP with feature “agreement” can be illustrated as following AVM:

cat	NP
agreement	[num sing per 3]

4.1.2 Feature path and reentrant structure

We can also use the DAG to represent the attribute-value pairs.

E,g above AVM can be represented by following DAG.

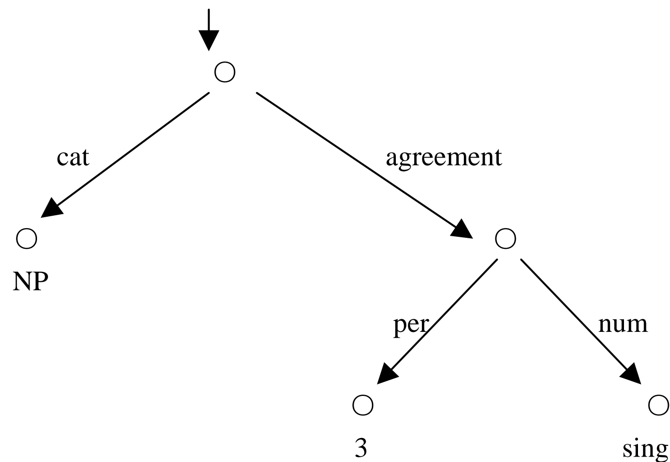


Fig. 1 DAG for feature structure

In DAG, a **feature path** is a list of features through a feature structure leading to a particular value. For example, in Fig. 1, we can say that the <agreement num> path leads to the value sing, <agreement per> path leads to the value 3.

If there is the shared feature structure, such feature structure will be referred to as **reentrant structure**. In the case of a reentrant structure, two feature paths actually lead to the same node in the structure.

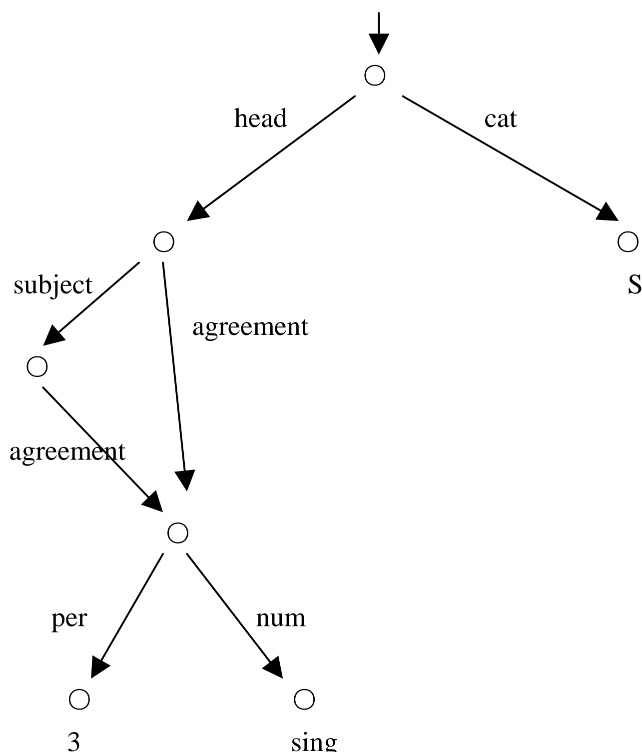


Fig. 2 a feature structure with shared values

In Fig. 2, the <head subject agreement> path and the <head agreement> path lead to the same location. They shared the feature structure

$$\begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix}$$

The shared structure will be denoted in the AVM by adding numerical indexes that signal the values to be shared.

$$\begin{bmatrix} \text{cat} & \text{s} \\ \text{head} & \begin{bmatrix} \text{agreement} & ① \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The reentrant structures give us the ability to express linguistic knowledge in the elegant ways.

4.1.3 Unification of feature structures

For the calculation of feature structure, we can use the unification to do it. There are two principle operations in the unification.:

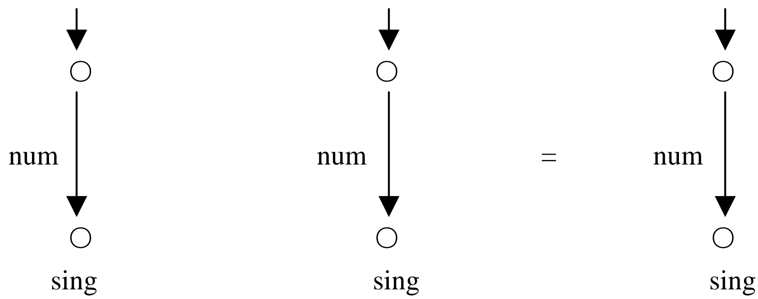
- Merging the information content of two structure that are compatible;
- Rejecting the merging of structures that are incompatible.

Following are the examples (symbol ∪ means unification):

(1) Compatible

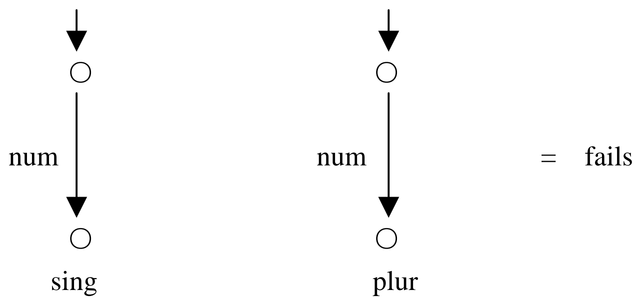
:

$$\begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \cup \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} = \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix}$$



(2) Incompatible:

[num sing] ∪ [num plur] = fails!

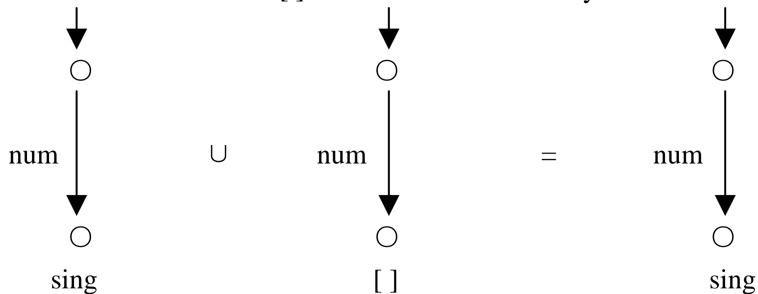


(3) Symbol []:

:

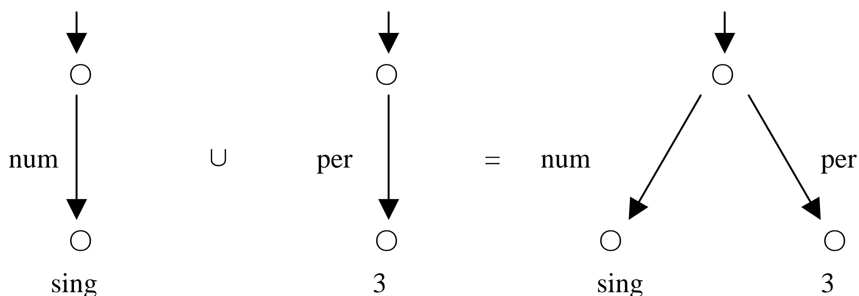
[num sing] ∪ [num [ ]] = [num sing]

The feature with a [ ] value can be successfully matched to any value.



(4) Merger:

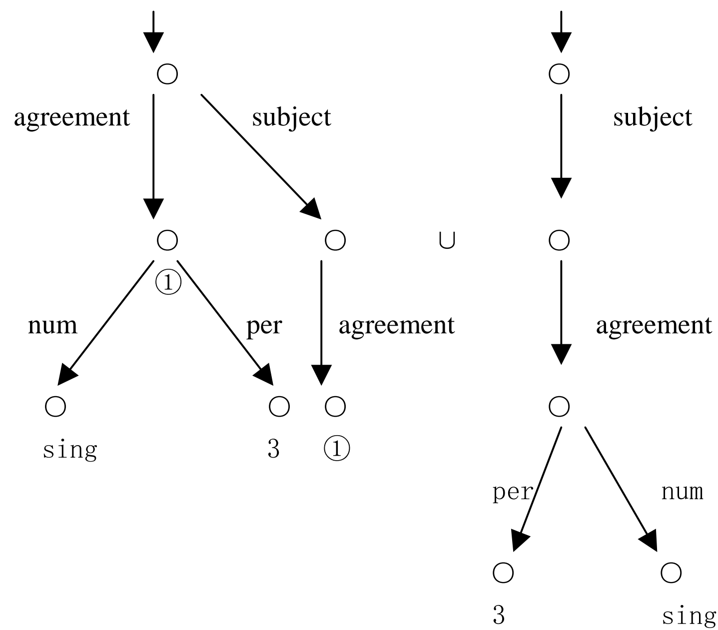
[num sing] ∪ [per 3] = [num sing / per 3]

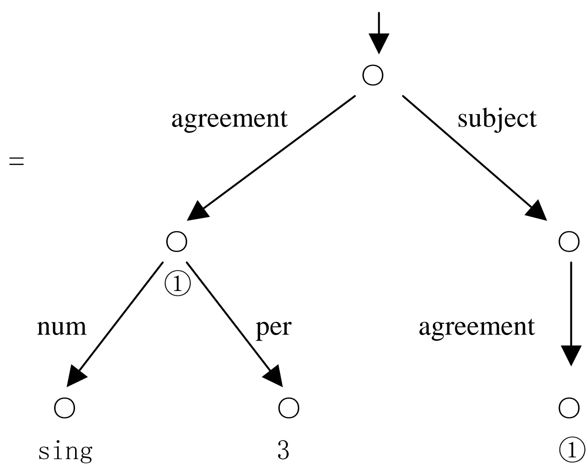


(5) The reentrant structure

$$\begin{bmatrix} \text{agreement} & ① \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix} \cup \begin{bmatrix} \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{agreement} & ① \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix}$$



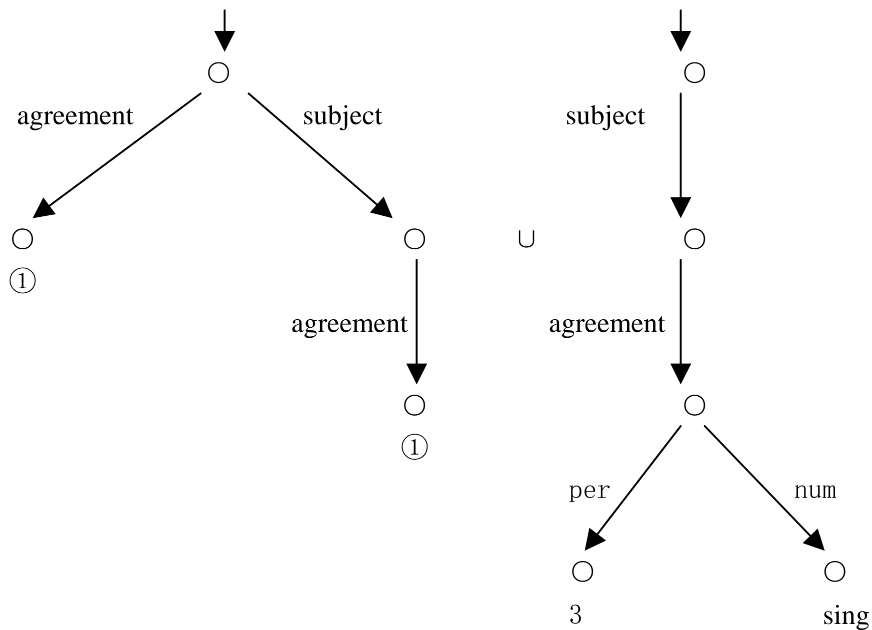


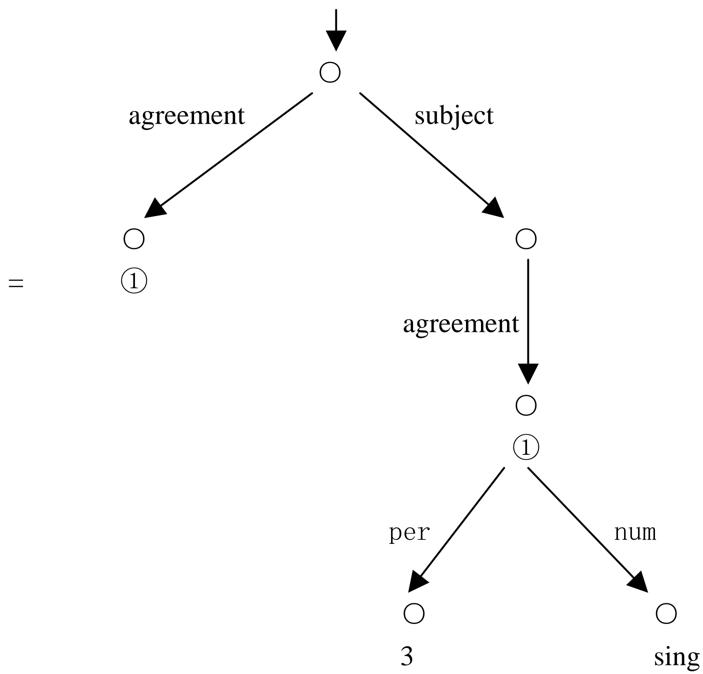
(5) The copying capability of unification

$$\begin{bmatrix} \text{agreement} & ① \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix}$$

$$\cup \begin{bmatrix} \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{agreement} & ① \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$





(5) The features merely have similar values:

In following example, there is no sharing index linking the “agreement” feature and [subject agreement], the information [per 3.] is not added to the value of the “agreement” feature.

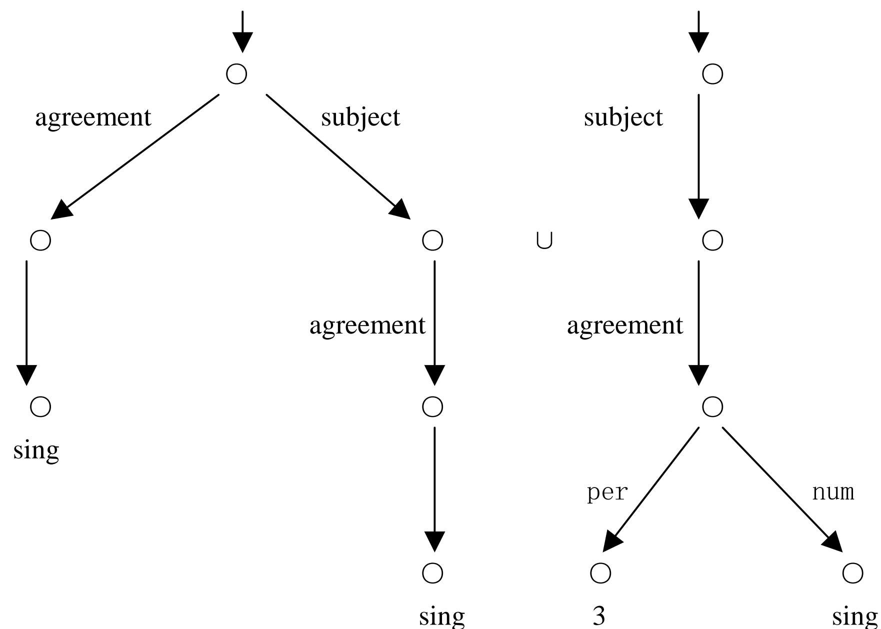
$$\begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

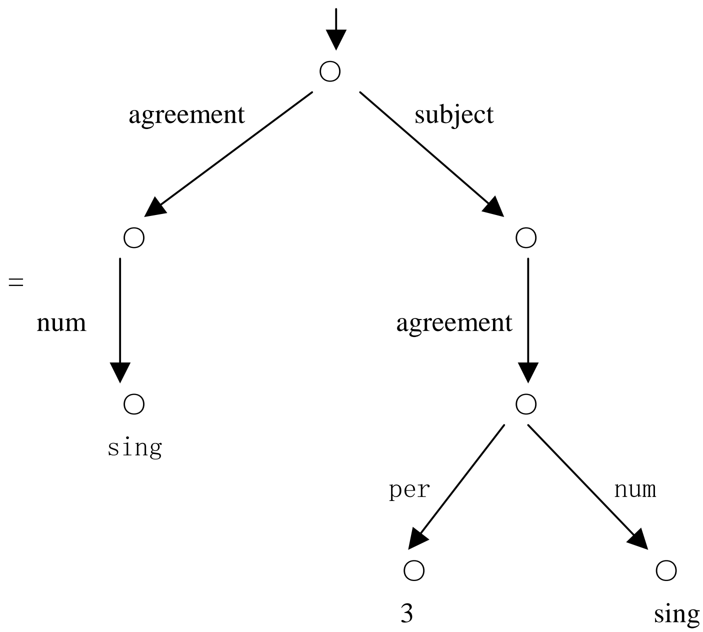
$$\cup \begin{bmatrix} \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

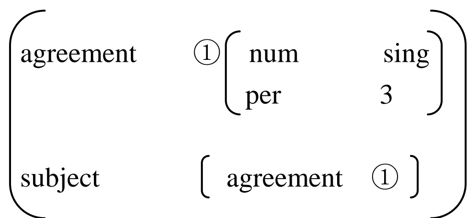
In the result, the information [per 3.] is only added to the end of [subject [agreement]] path, but it is not added to the end of “agreement’ (it is first line in the AVM of result). Therefore, the value of “agreement” is only [num sing] without [per 3].

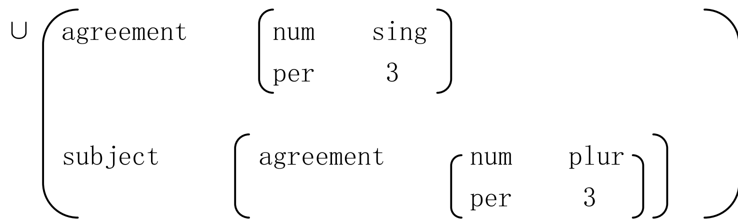
.



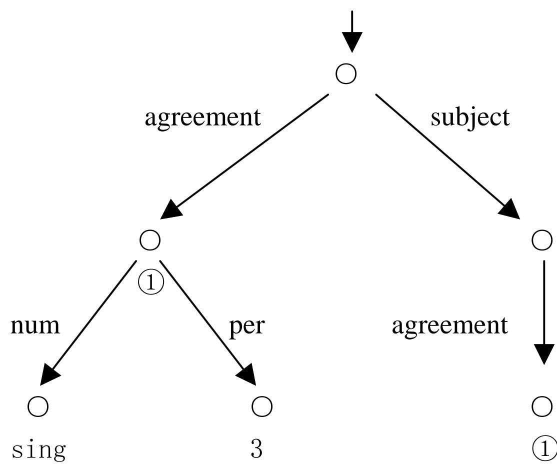


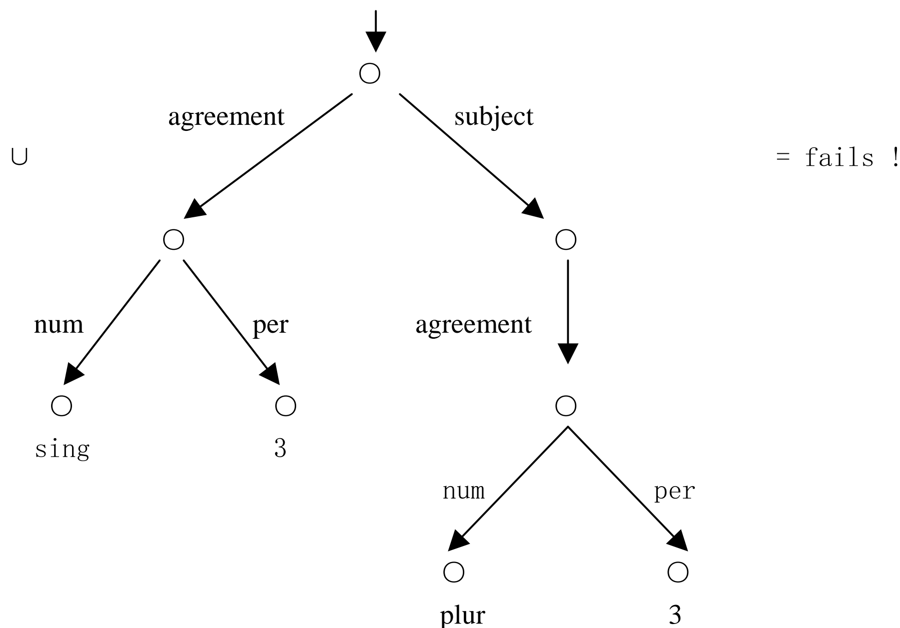
(7) The failure of unification





= fails !





Feature structures are a way of representing partial information about some linguistic object or placing informational constrains on what the object can be. Unification can be seen as a way of merging the information in each feature structure, or describing objects that satisfy both sets of constraints.

4.1.4 Subsumption

Intuitively, unifying two feature structures produces a new feature structure that is more specific (has more information) than, or is identical to, either of the input feature structure. We say that a less specific (more abstract) feature structure subsumes an equally or more specific one.

Formally, A feature structure F subsumes a feature structure G if and only if:

- For every feature x in F, F(x) subsumes G(x) (where F(x) means “the value of the feature x of 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).

E.g.

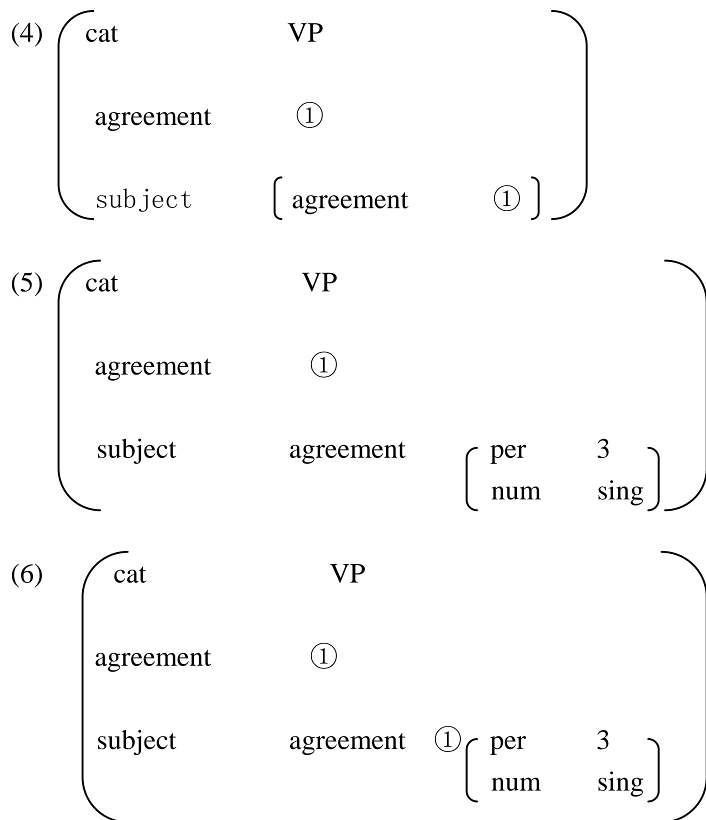
(1) $\begin{bmatrix} \text{num} & \text{sing} \end{bmatrix}$

(2). $\begin{bmatrix} \text{per} & 3 \end{bmatrix}$

(3) $\begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix}$

We have: (1) subsumes (3),

(2) subsumes (3).



We have: (3) subsumes (5), (4) subsumes (5), (5) subsumes (6), (4) and (5) subsume (6)..

Subsumption is a partial ordering: there are pairs of feature structures that neither subsume nor are subsumed by each other:

(1) does not subsume (2),
(2) does not subsume (1),
(3) does not subsume (4),
(4) does not subsume (3).

Since every feature structure is subsumed by the empty structure [], the relation among feature structures can be defined as a semi-lattice. The semi-lattice is often represented pictorially with the most general feature [ ] at the top and the subsumption relation represented by lines between feature structures.

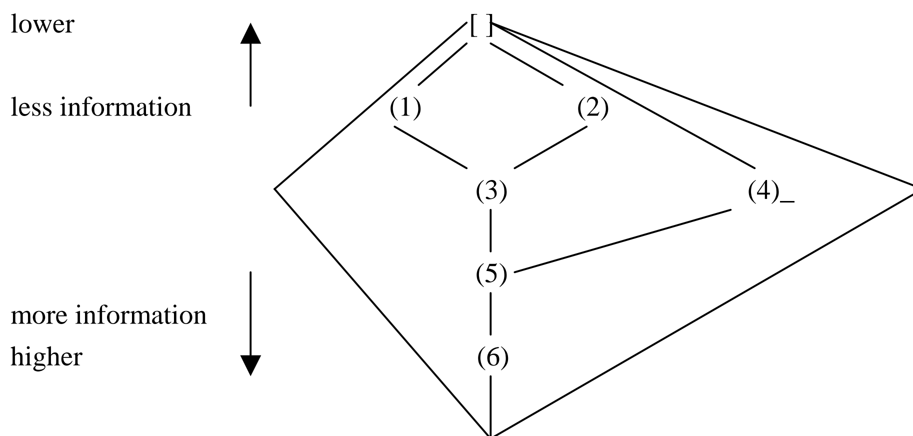


Fig. 4 subsumption represented by semi-lattice

.

4.1.5 Formal definition of Unification

Unification can be formally defined in terms of the subsumption semi-lattice as follows:.

Given two feature structures F and G, the unification F∪G is defined as the most general feature H such that F subsume H and G subsume H.

Since the information ordering defined by unification is a semi-lattice, the unification operation is monotonic. This means:

- If some description is true of a feature structure, unifying it with another feature structure results in a feature structure that still satisfies the original description.
- The unification operation is order-independent; given a set of feature structures to unify, we can check them in any order and get the same result

Unification is a way of implementing the integration of knowledge from different constraints:

- Given two compatible feature structures as input, it produces a new feature structure which contains all the information in the inputs;
- Given two incompatible feature structures, it fails.

4.2 Feature structures in the Grammar

4.2.1 Augmentation of CFG rules with feature structures:

- To associate complex 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 constructions.

Formally, we can use following notation to denote the grammar augmentation:

$\beta_0 \rightarrow \beta_1 \cdots \beta_n$

(set of constraints)

The specified constraints have one of the following forms:

(β_i feature path) = Atomic value

(β_i feature path) = (β_j feature path)

The notation (β_i feature path) denotes a feature path through the feature structure associated with the β_i component of the CFG rule.

For example, the rule

S → NP VP

can be augmented with attachment of the feature structure for number agreement as follows:

S → NP VP

(NP num) = (VP num)

In this case, the simple generative nature of CFG rule has been fundamentally changed by this augmentation. These changes are following two aspects:

- The elements of CFG rules will have feature-based constraints associated with them. This reflects a shift from atomic grammatical categories to more complex categories with properties.
- The constraints associated with individual rules can refer to the feature structures associated with the parts of the rule to which they are attached.

4.2.2 Agreement

There are two kinds of agreement in English.

- Subject-verb agreement

  S → NP VP

  (NP agreement) = (VP agreement)

  E.g. This flight serves breakfast.

  These flights serve breakfast.

  S → Aux NP VP

  (Aux agreement) = (NP agreement)

  E.g. Does this flight serve breakfast?

  Do these flights serve breakfast?

- Determiner-nominal agreement

  NP → Det Nominal

  (Det Agreement) = (Nominal Agreement)

  (NP Agreement) = (Nominal Agreement)

  E,g. This flight.

  These flights.

The constraints involve both lexical and non-lexical constituents.

- The constraints of lexical constraints can directly write in the lexicon:

Aux → do

(Aux agreement num) = plur

(Aux agreement per) = 3

Aux → does

(Aus agreement num) = sing

(Aux agreement per) = 3

Determiner → this

(Det agreement num) = sing

Determiner → these

(Det agreement num) = plur

Verb → serves

(Verb agreement num) = sing

Verb → serve

(Verb agreement num) = plur

Noun → flight

(Noun agreement num) = sing

Noun → flights

(Noun agreement num) = plur

- The constraints of non-lexical constituent can acquire values for at least some of their features from their component constituents.

VP → Verb NP

(VP agreement) = (Verb agreement)

The constraints of ”VP” come from the constraints of “Verb”.

Nominal → Noun

(Nominal agreement) = (Noun agreement)

The constraints of “Nominal” come from the “Noun”.

4.2.3 Head features

The features for most grammatical categories are copied from one of the children to the parent. The child that provides the feature is called **the head of the phrase**, and the features copied are called **head features**.

In the following rules,

VP → **Verb** NP

(VP agreement) = (**Verb** agreement)

NP → Det **Nominal**

(Det agreement) = (**Nominal** agreement)

(NP agreement) = (**Nominal** agreement)

Nominal → **Noun**

(Nominal agreement) = (**Noun** agreement)

the verb is the head of the VP, the nominal is the head of NP, the Noun is the head of the nominal. In these rules, the constituent providing the agreement feature structure up to the parent is the head of the phrase. We can say that the agreement feature structure is a head feature.

We can rewrite our rules by placing the agreement feature structure under a HEAD feature and then copying that feature upward:

VP → **Verb** NP

(VP head) = (**Verb** head)

NP → Det **Nominal**

(Det head Agreement) = (**Nominal** head Agreement)

Det and Nominal locate in the same level, their “HEAD Agreement” is equal.

(NP head) = (**Nominal** head)

Nominal → **Noun**

(Nominal head) = (**Noun** head)

The lexical rules can be rewritten as follows:

Verb → serves

(Verb head agreement num) = sing

Verb → serve

(Verb head agreement num) = plur

Noun → flight

(Noun head agreement num) = sing

Noun → flights

(Noun head agreement num) = plur

The conception of a head is very significant in grammar, because it provides a way for a syntactic rule to be linked to a particular word.

4.2.4 Sub-categorization

4.2.4.1 An atomic feature SUBCAT:

Following is a rule with complex features

Verb-with-S-comp → think

VP → Verb-with-S-comp S

We have to subcategorize the verbs to some subcategories. So we need an atomic feature called SUBCAT.

- Opaque approach

  Lexicon:

  Verb → serves

  <Verb head agreement num> = sing

  <Verb head subcat> = trans

  Rules:

  VP → Verb

  <VP head> = <Verb head>

  <VP head subcat> = intrans

  VP → Verb NP

  <VP head> = <Verb head>

  <VP head subcat> = trans

  VP → Verb NP NP

  <VP head> = <Verb head.

  <VP head subcat. = ditrans

In these rules, the value of SUBCAT is un-analyzable. It does not directly encode either the number or type of the arguments that the verb expects to take.

This approach is somewhat opaque, it is not so clear.

- .Elegant approach:

  A more elegant approach makes better use of the expressive power of feature structures, allows the verb entries to directly specify the order and category type of the arguments they require.

  The verb's subcategory feature expresses a list of its objects and complements.

  Lexicon:

  Verb → serves

  <Verb head agreement num> = sing

<Verb head subcat first cat> = NP
<Verb head subcat second> = end

Verb → leaves

<Verb head agreement num> = sing
<Verb head subcat first cat> = NP
<Verb head subcat second cat> = PP
<Verb head subcat third> = end

E..g. “we leave Seoul in the morning”.

Rules:

VP → Verb NP

<VP head> = <Verb head>
<VP head subcat first cat> = <NP cat>
<VP head subcat second> = end

4.2.4.2 Sub-categorization frame

The sub-categorization frame can be composed of many different phrase types.

- Sub-categorization of verb:

Each verb allows many different sub-categorization frames. For example, verb ‘ask’ can allow following sub-categorization frame:

Subcat:	Example
Quo .	asked [$_{Quo}$ “What was it like?”]
NP	asking [$_{NP}$ a question]
Swh	asked [$_{Swh}$ what trades you’re interested in]
Sto	ask [$_{Sto}$ him to tell you]
PP	that means asking [$_{PP}$ at home]
Vto	asked [$_{Vto}$ to see a girl called Sabina]
NP Sif	asked [$_{NP}$ him] [$_{Sif}$ whether he could make]
NP NP	asked [$_{NP}$ myself] [$_{NP}$ a question]
NP Swh	asked [$_{NP}$ him [$_{Swh}$ why he took time off]

A number of comprehensive sub-categorization frame tagsets exist. For example, COMLEX (Macleod, 1998), ACQUILEX (Sanfilippo, 1993).

- Sub-categorization of Adjective

Subcat:	Example
Sfin	It was **apparent** [$_{Sfin}$ that the kitchen was the only room…]
PP	It was **apparent** [$_{PP}$ from the way she rested her hand over his]
Swheth	It is **unimportant** [$_{Swheth}$ whether only a little bit is accepted]

- Sub-categorization of noun

Subcat:	Example
Sfin	the **assumption** [$_{Sfin}$ that wasteful methods have been employed]
Swheth	the **question** [$_{Swheth}$ whether the authorities might have decided]

4.2.5 Long-Distance Dependencies

Sometimes, a constituent subcategorized for by the verb is not locally instantiated ,but is in a long-distance relationship with the predicate.

For example, following sentence:

Which flight do you want me to have the travel agent book?

Here, "which flight" is the object of "book", there is a long-distance dependency between them.

The representation of such long-distance dependency is a very difficult problem, because the verb whose subcategorization requirement is being filled can be quite distance from the filler.

Many solutions to representing long-distance dependency were proposed in unification grammars. One solution is called "Gap List". The gap list implements a list as a feature gap, which is passed up from phrase to phrase in the parse tree. The filler (E.g. "which flights") is put in the gap list, and must eventually be united with the subcategorization frame of some verb.

4.3 Implementing unification

4.3.1 Unification data structures

The unification operator takes two feature structures as input and returns a single merged feature if successful, or a feature signal if the two inputs are not compatible. The implementation of the operator is a relatively straightforward recursive graph matching algorithm. The algorithm loops through the features in one input and attempts to find a corresponding feature in the other. If all of feature match, then the unification is successful. If any single feature causes a mismatch then the unification fails.

The feature structures are represented using DAGs with additional fields. Each feature structure consists of two fields:

- A content field:
- A pointer field.

The content field may be null or contain a pointer to another feature structure. Similarly, the pointer field may be null or contain a pointer to another feature structure.

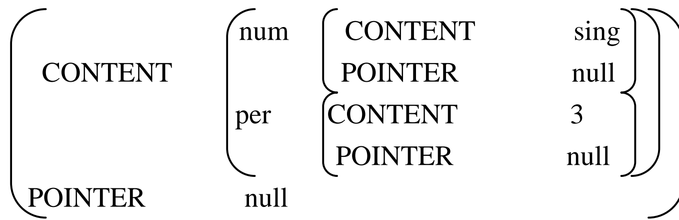
The operation is as follows:

- If the pointer field of the DAG is null, then the content field of the DAG contains the actual feature structure to be processed.
- If the pointer field is non-null, then the destination of the pointer represents the actual feature structure to be processed.
- The merger aspects of unification will be achieved by altering the pointer field of DAGs during processing.

For example, if we have the following feature structure:

$$\begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix}$$

The extended DAG representation is as following:



The DAG is as follows:

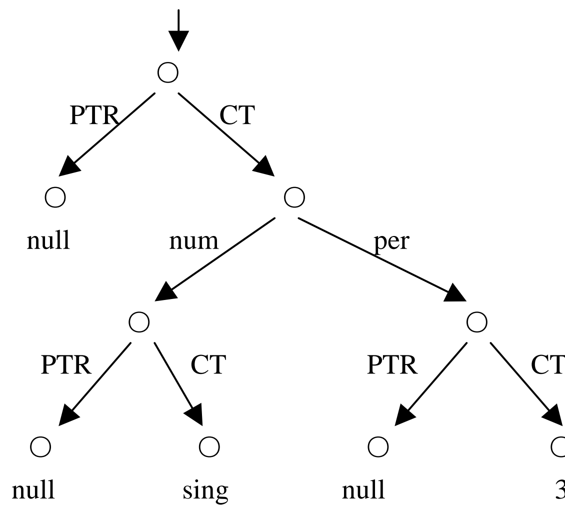


Fig. 5. An extended DAG notation

The example of the unification of feature structures is as follows:

$$\begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \cup \begin{bmatrix} \text{per} & 3 \end{bmatrix} = \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix}$$

The DAGs of original arguments is as follows:

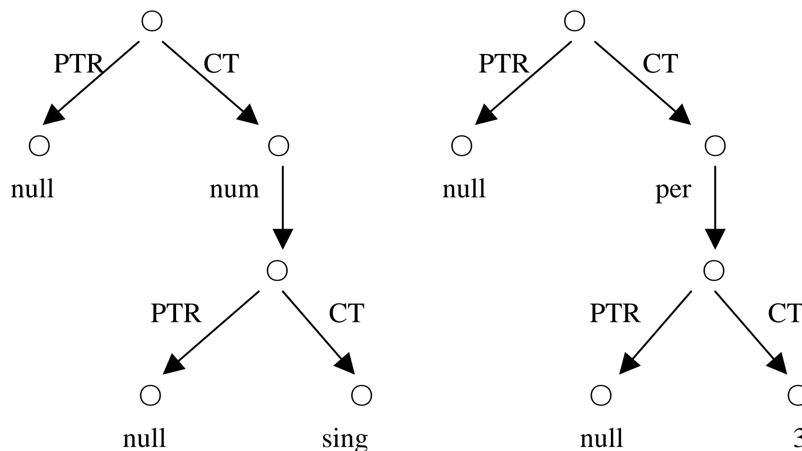


Fig. 6 The original arguments

The unification shall result in the creation of a new structure containing the union of the information from the two original arguments:

- Adding a "per" feature to the first argument;
- Assigning it a value by filling its PTR field with a pointer to the appropriate location in the second argument.

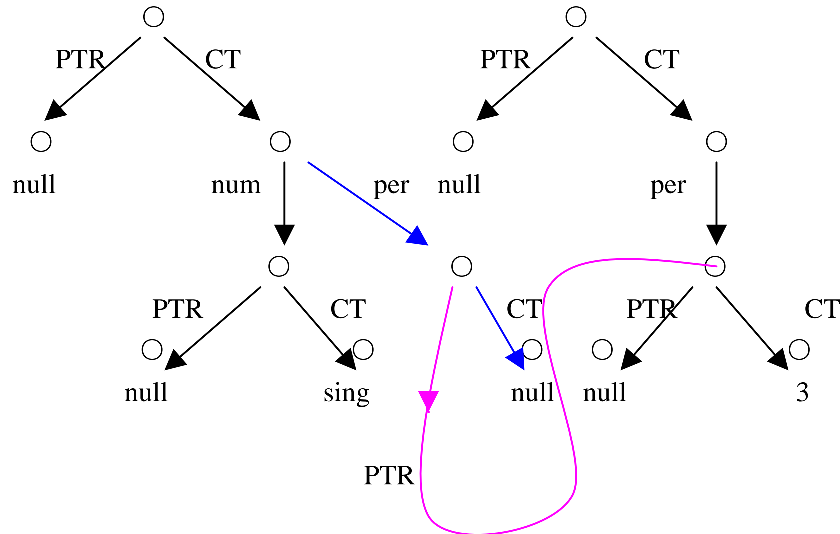


Fig.7. adding a "per' feature

- Set the pointer field of the second argument to point at the first one.

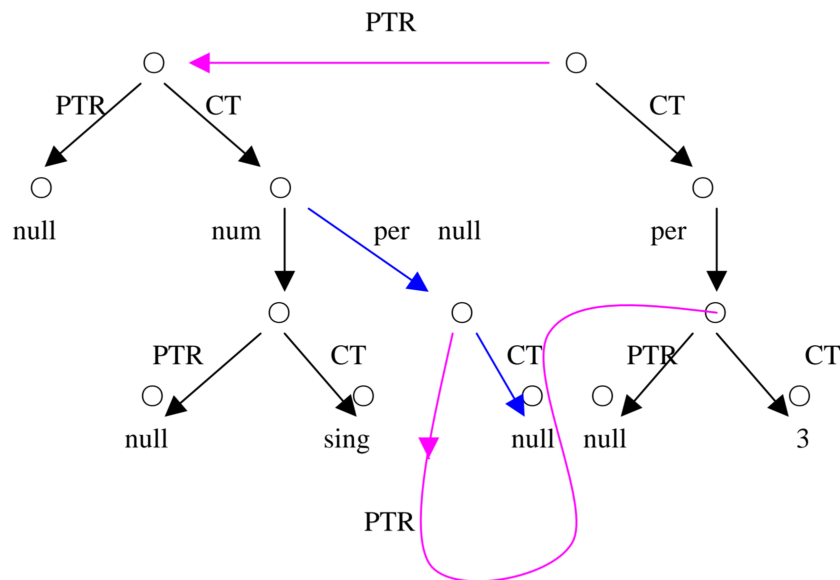
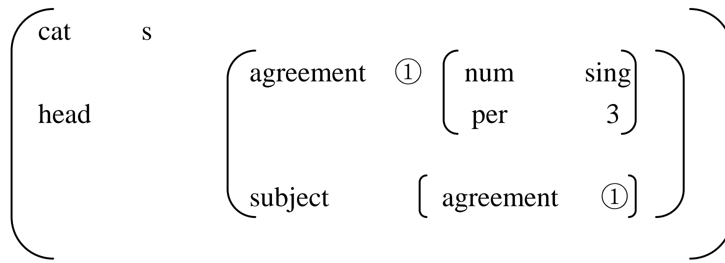


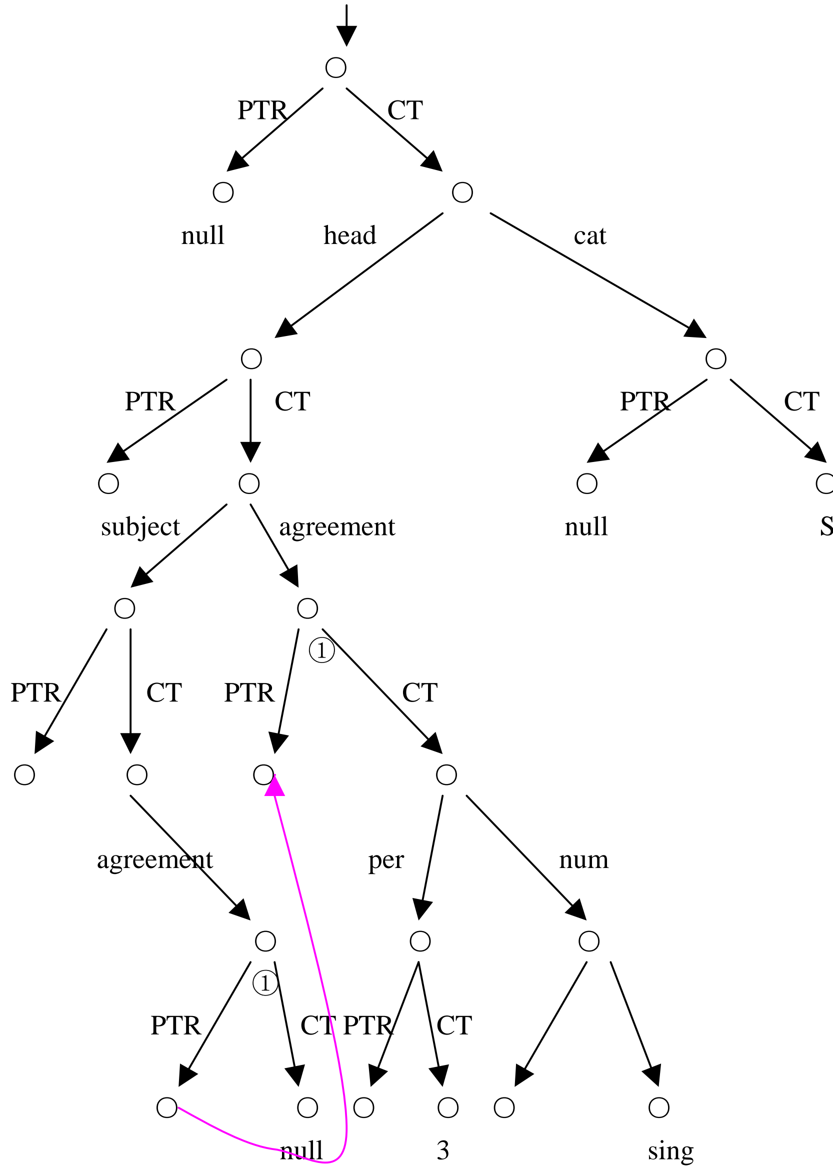
Fig.8. The final result of unification

More complex examples:

(1) The **reentrant structure**.



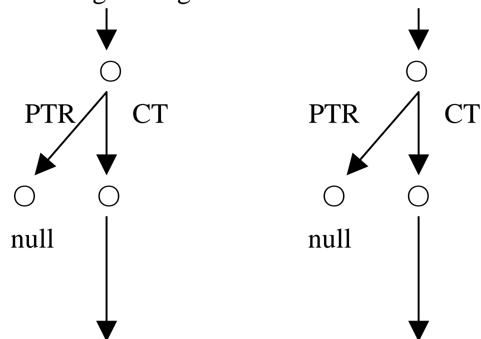
The DAG:

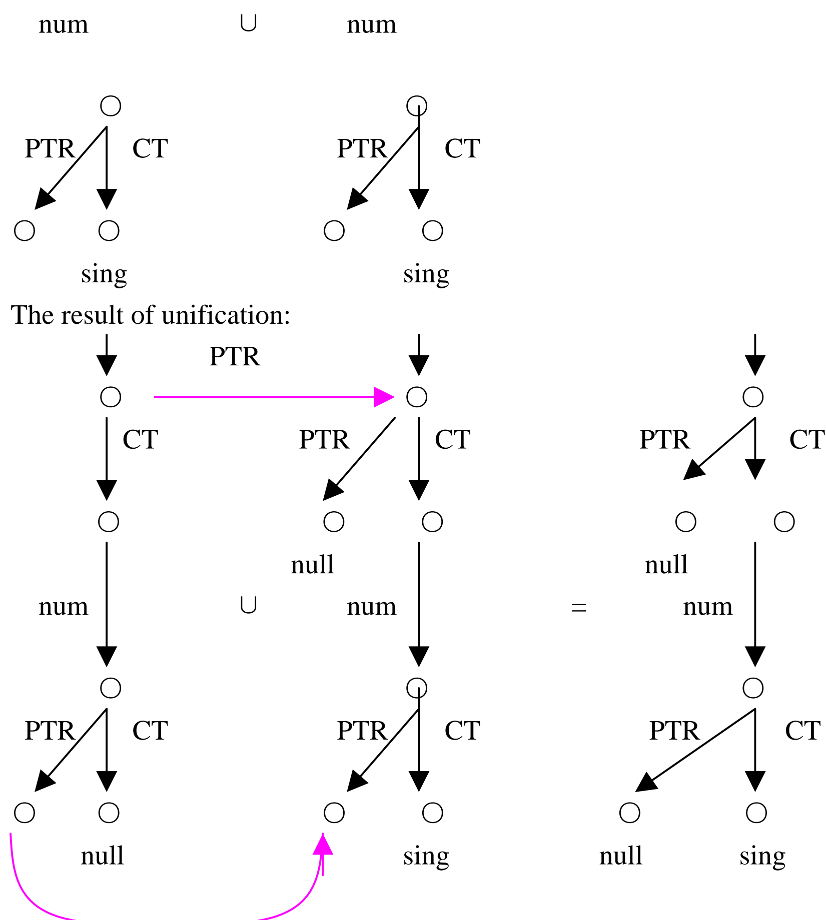


(2) Compatible feature structure:

$[\text{num} \quad \text{sing}] \cup [\text{num} \quad \text{sing}] = [\text{num} \quad \text{sing}]$

The original arguments::



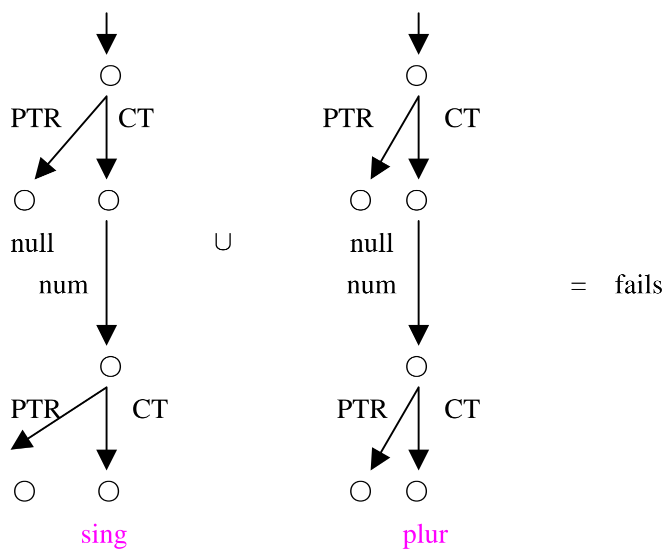


The result of unification:

(3) Incompatible:

$\begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \cup \begin{bmatrix} \text{num} & \text{plur} \end{bmatrix} = \text{fails!}$

The result of unification:

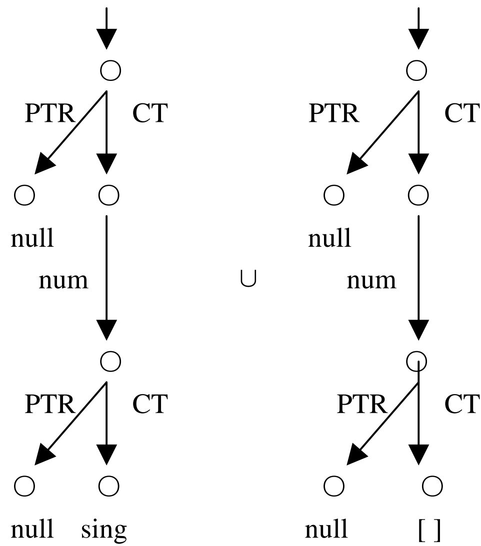


(4) Symbol []:

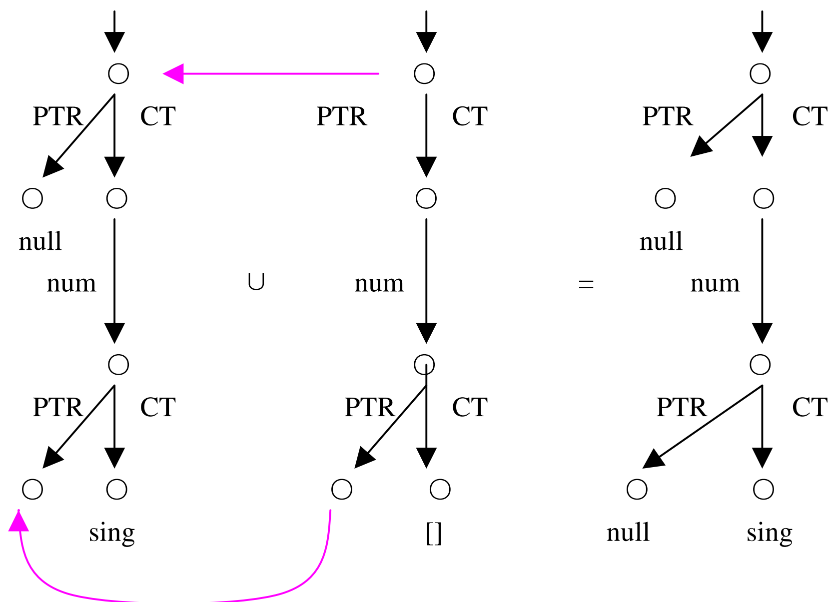
:

$$\begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \cup \begin{bmatrix} \text{num} & [\,] \end{bmatrix} = \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix}$$

The original arguments:



The result of unification:

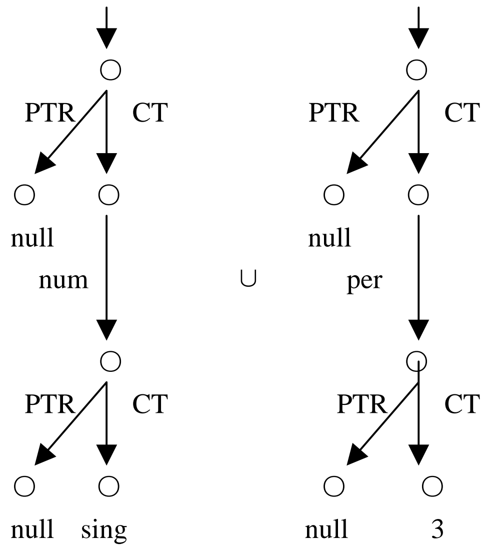


(5) Merger

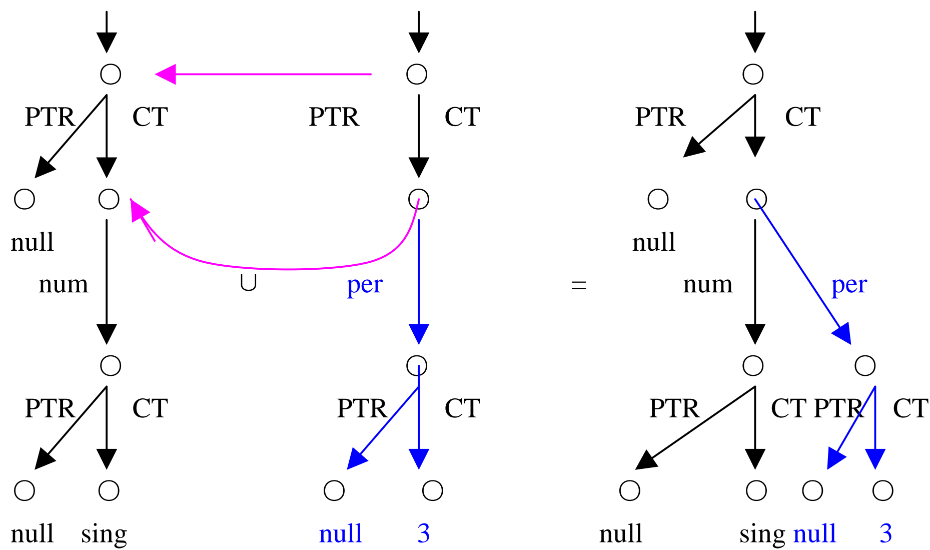
:

$$\begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \cup \begin{bmatrix} \text{per} & 3 \end{bmatrix} = \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix}$$

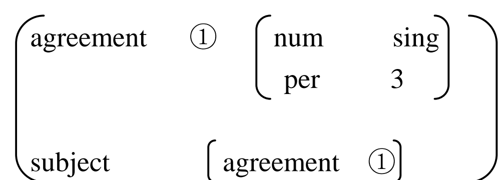
The original arguments:



The result of unification:



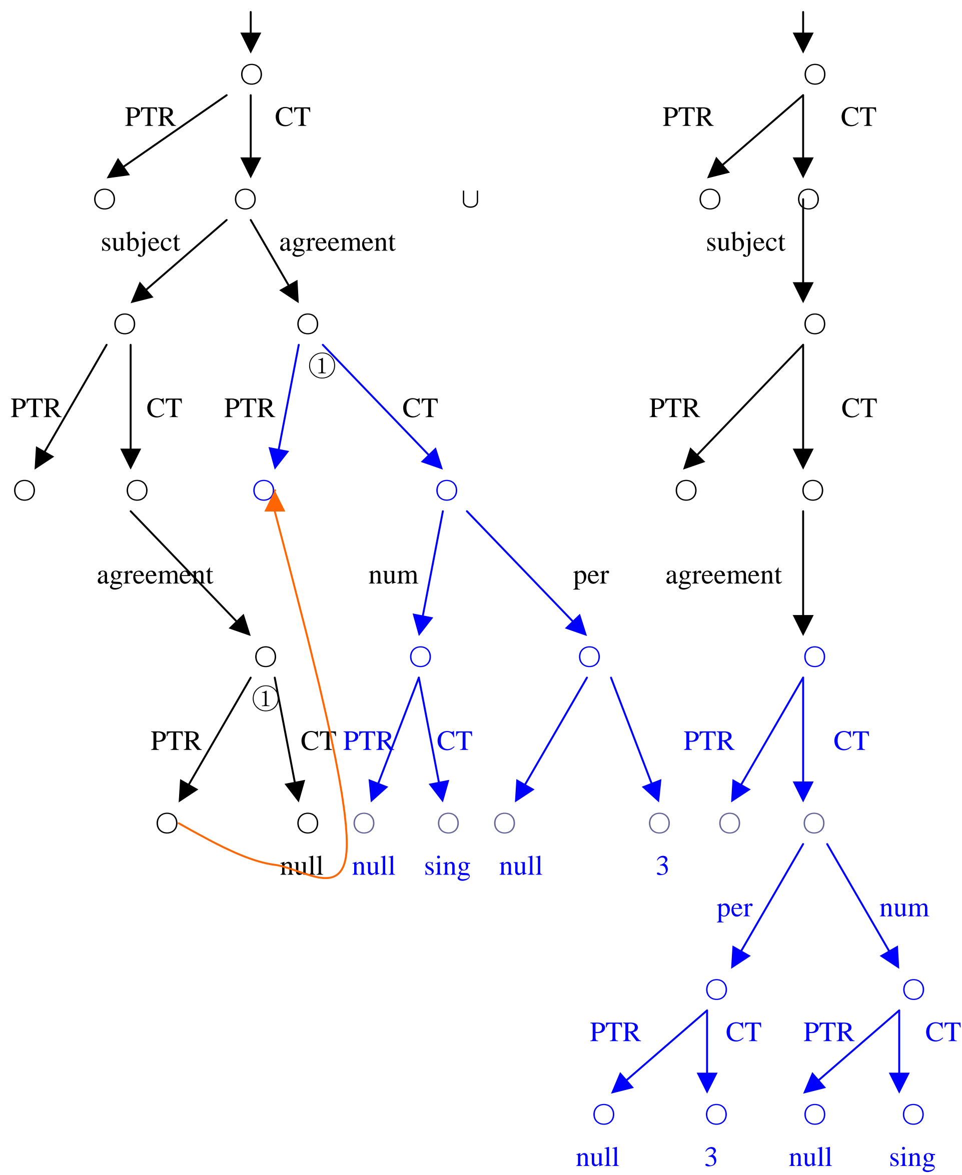
(6) The reentrant structure



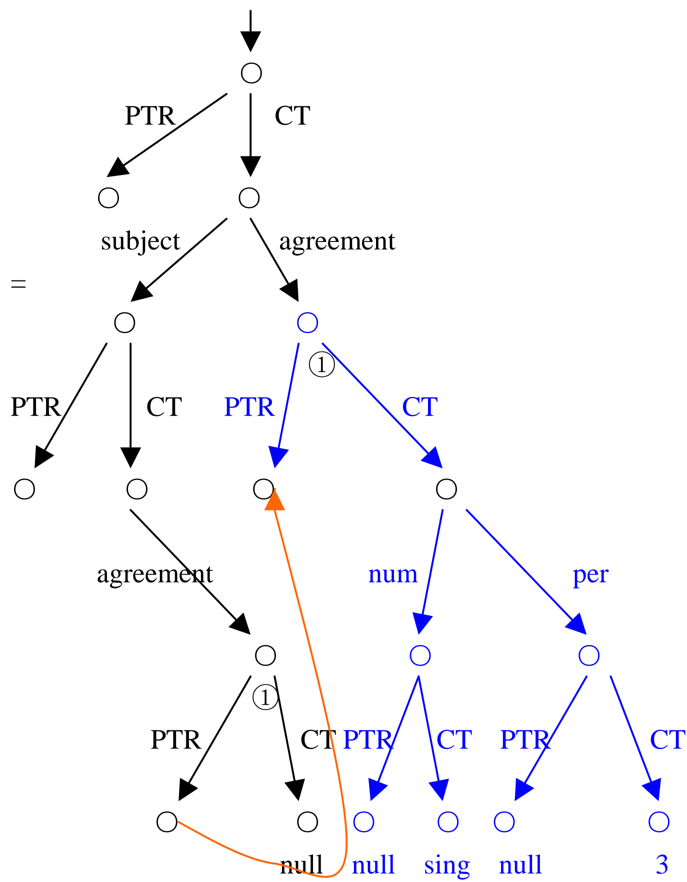
$$\cup \begin{bmatrix} \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{agreement} & ① \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix}$$

The original arguments:



The result of unification:



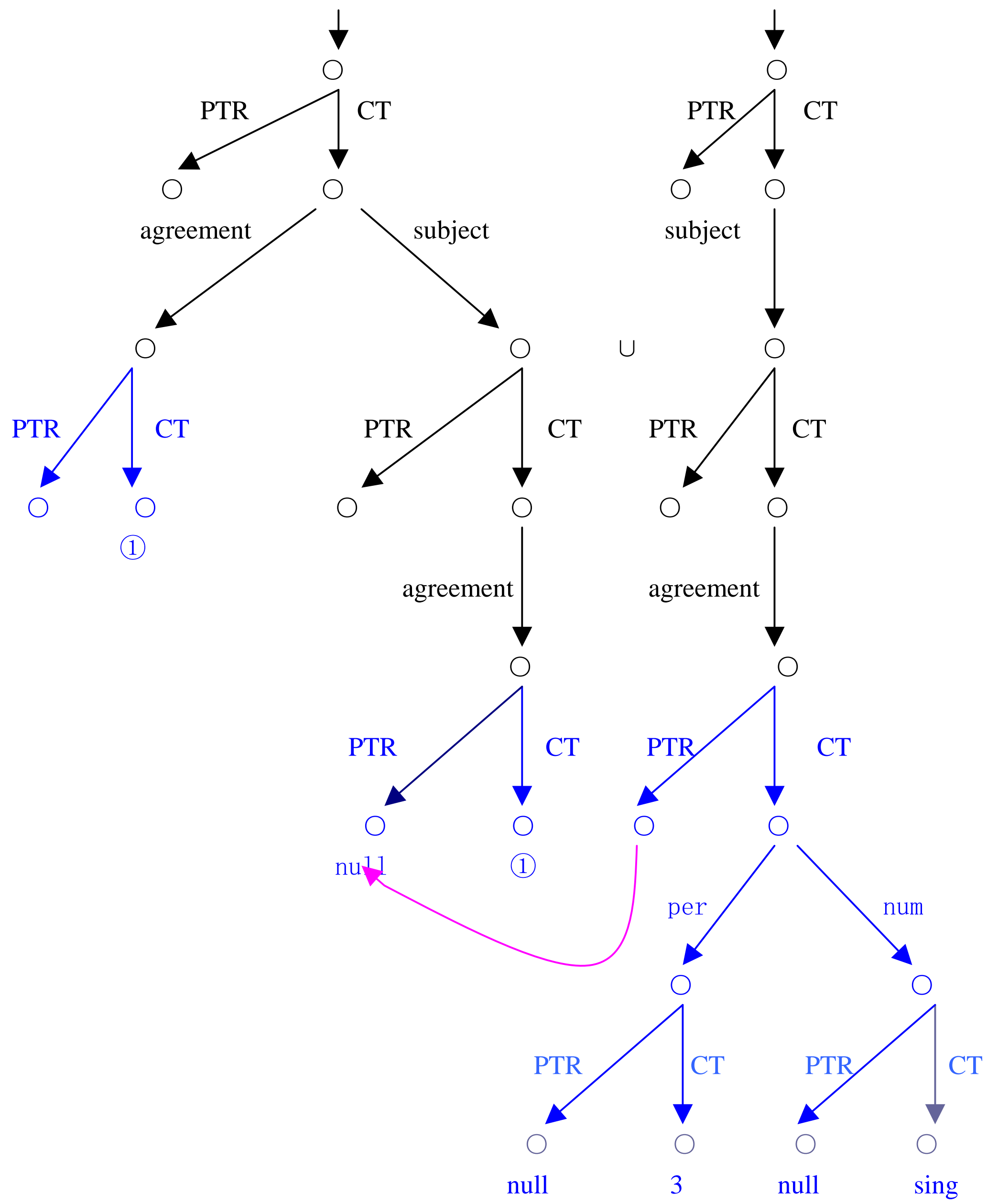
(7) The copying capability of unification

$$\begin{bmatrix} \text{agreement} & ① \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix}$$

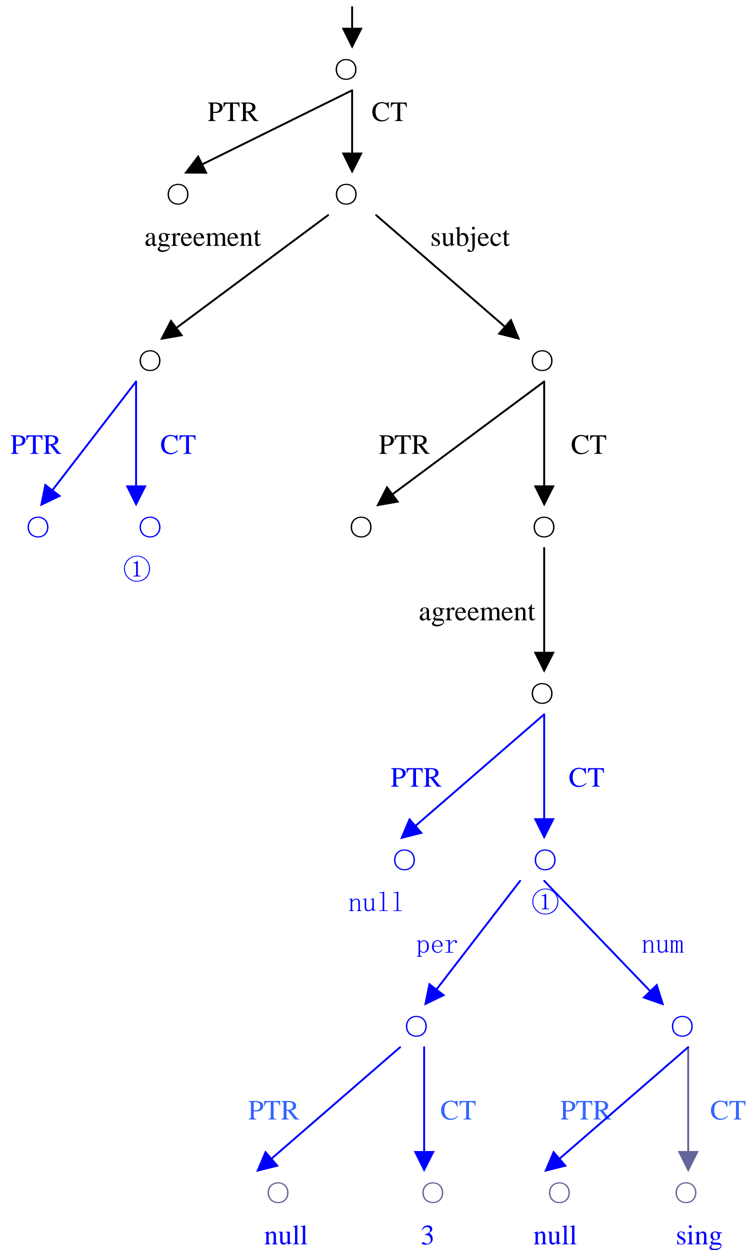
$$\cup \begin{bmatrix} \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{agreement} & ① \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The original arguments:



The results of unification:



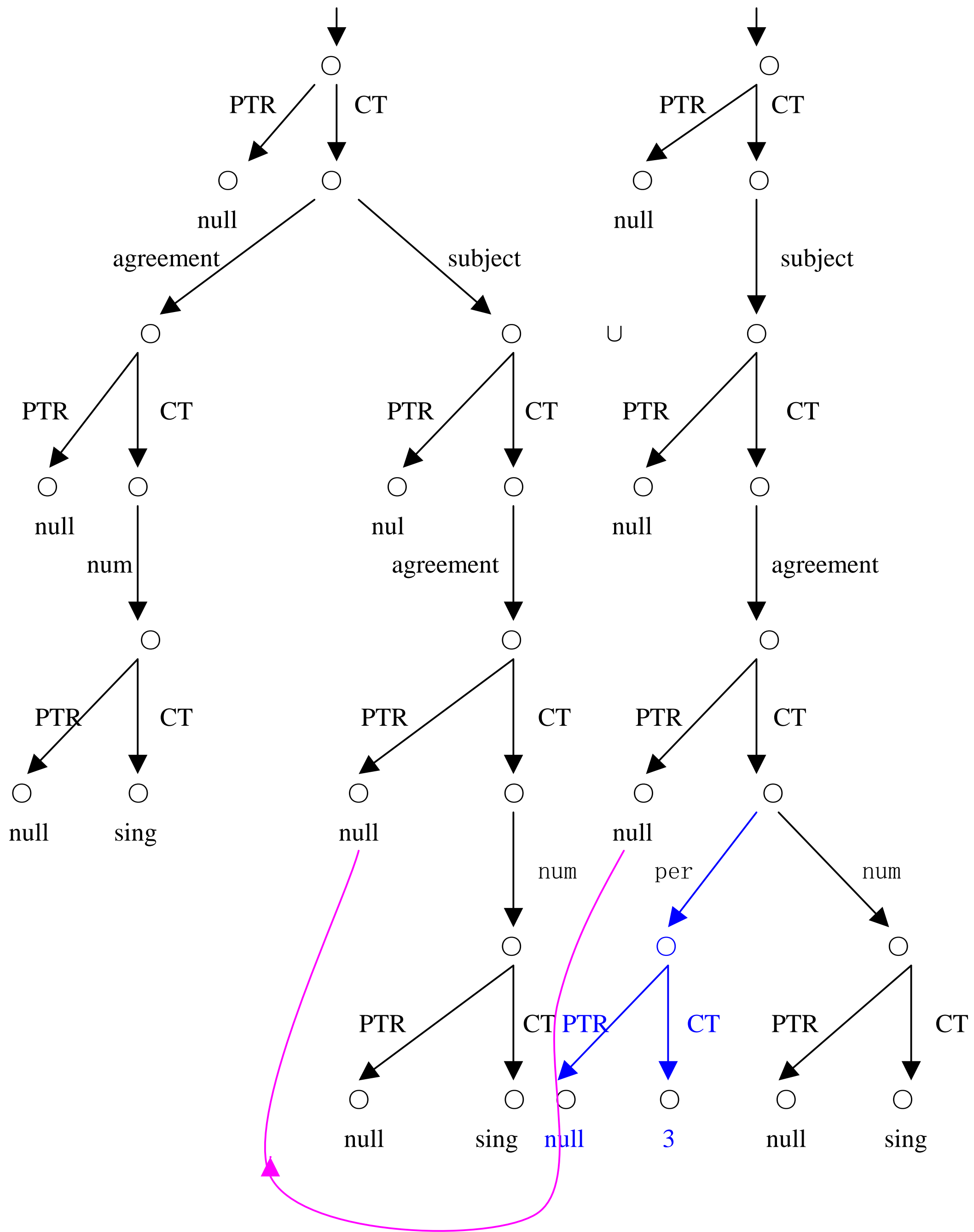
(8) The features merely have similar values:

$$\begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

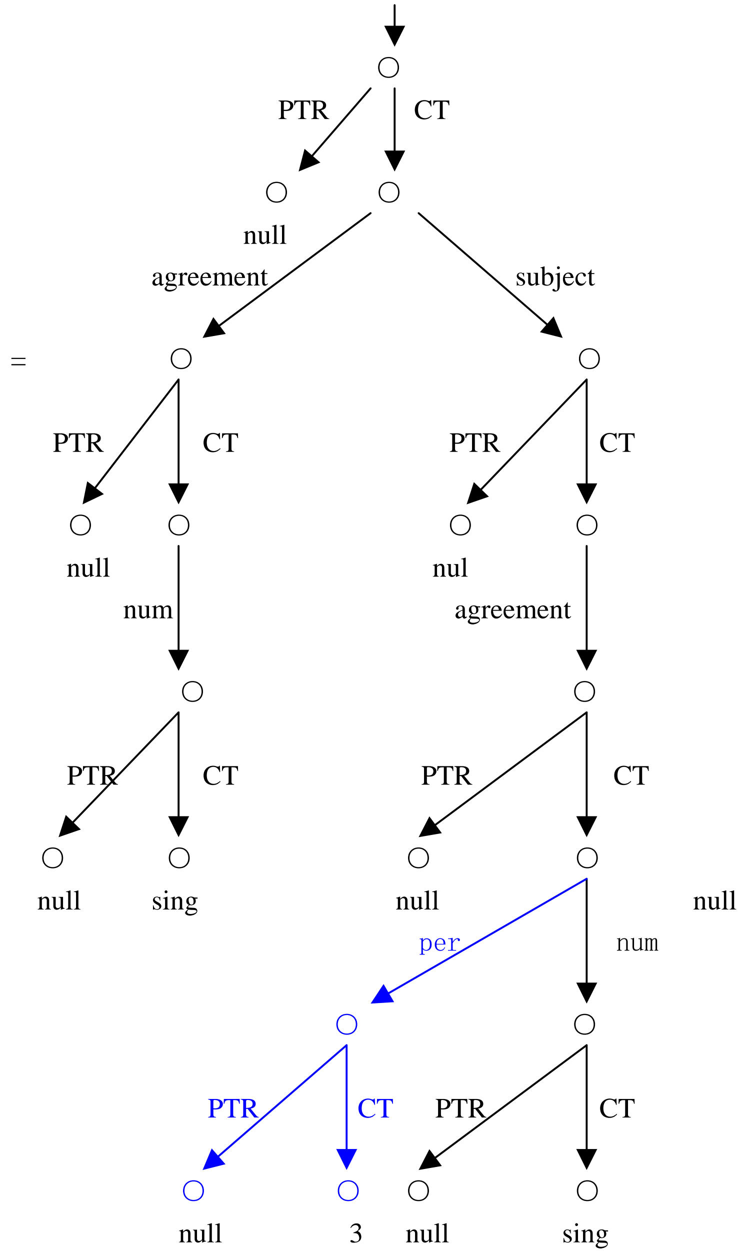
$$\cup \begin{bmatrix} \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \\ \text{num} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The original arguments:



The results of unification:



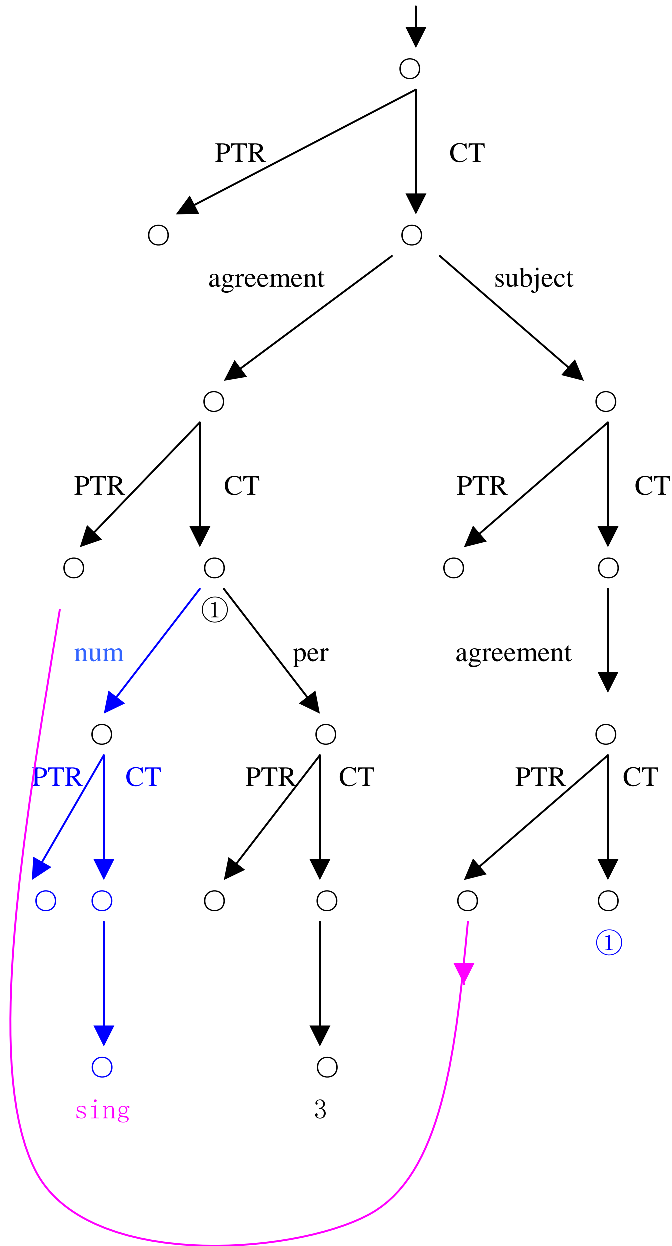
(9) The failure of unification

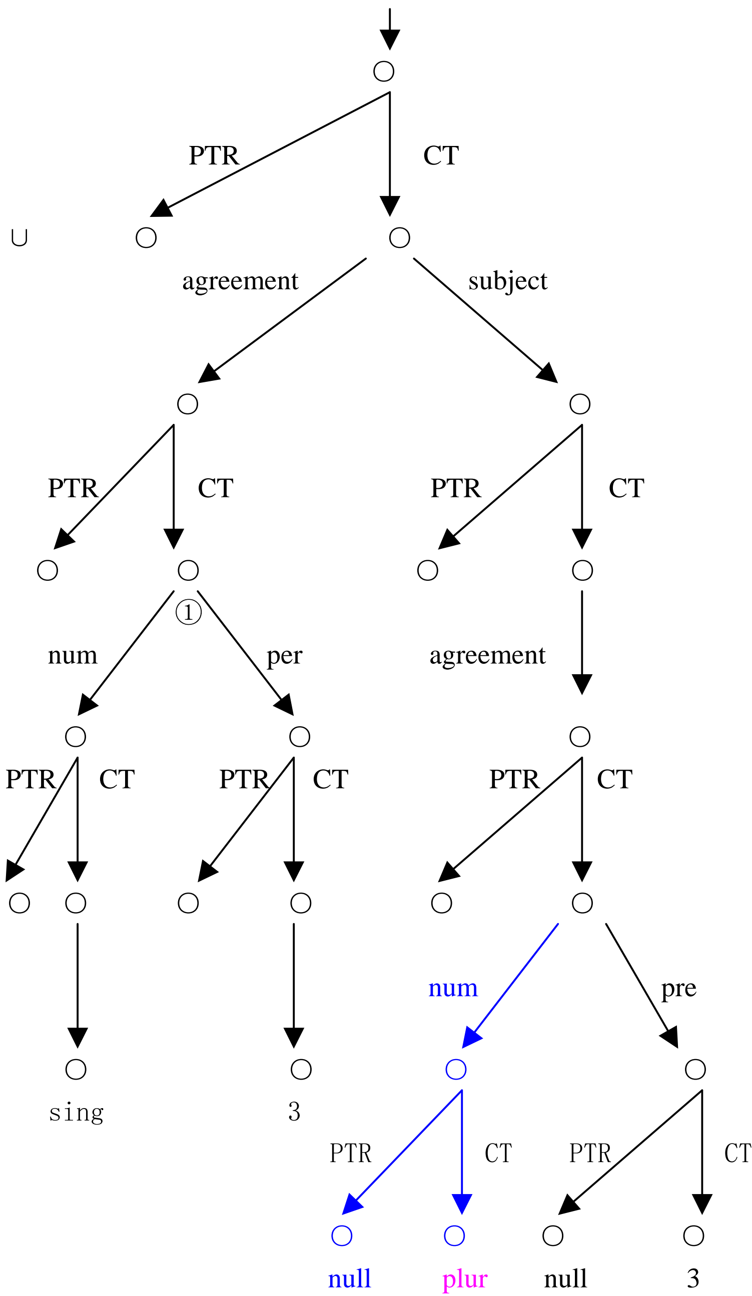
$$\begin{bmatrix} \text{agreement} & ①\begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \\ \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix}$$

$$\cup \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & 3 \end{bmatrix} \\ \\ \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{num} & \text{plur} \\ \text{per} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

= fails !

The original arguments:





= fails !

4.3.2 The unification Algorithm

The unification algorithm is as follows:

```
function UNIFY (f1, f2) returns fstructure or failure
f1-real ← Real contents of f1
f2-real ← Real contents of f2
If f1-real is null then
    f1.pointer ← f2
    return f2
else if f2-real is null then
    f2.pointer ← f1
    return f1
else if f1-real and f2-real are identical then
    f1.pointer ← f2
    return f2
else if both f1-real and f2-real are complex feature structure then
    f2.pointer ← f1
    for each feature in f2-real do
        other feature ← Find or create
                        a feature corresponding to feature in f1-real
        if UNIFY (featurer.value, other feature.value) returns failure then
            return failure
    return f1
else return failure
```

"←" means "be changed to point to" or "be set to".

- First step: To acquire the true contents of both of the arguments.. The valuables f1-real and f2-real are the result of this pointer following process.
- Second step: To test for the various base cases of the recursion. There are three possible base cases:
    1. One or both of the arguments has a null value;
    2. The arguments are identical;
    3. The arguments are non-complex and non-identical.
- In the case where either of the arguments is null, the pointer field for the null argument is changed to point to the other argument, which is then returned. The result is the both structures now point at the same value.
- If the structure are identical, then the pointer of the first is set to the second and the second is returned.
- If neither of the preceding tests is true, then there are two possibilities: they are non-identical atomic values, or they are non-identical complex structures. The former case signals an incompatibility in the arguments that leads the algorithm to return a failure signal. In the latter case, a recursive call is needed to ensure that the component parts of the complex structures are compatible. In this implementation, the key to the recursion is a loop over all the features of the second argument (f2). This loop attempts to unify the value of each feature in f2 with the corresponding feature in f1. In this loop, if a feature is encountered in f2 that is missing from f1, a feature is added to f1 and given the value NULL. Processing then continues as if the feature had been there to begin with. If every one of these unifications succeeds, then the

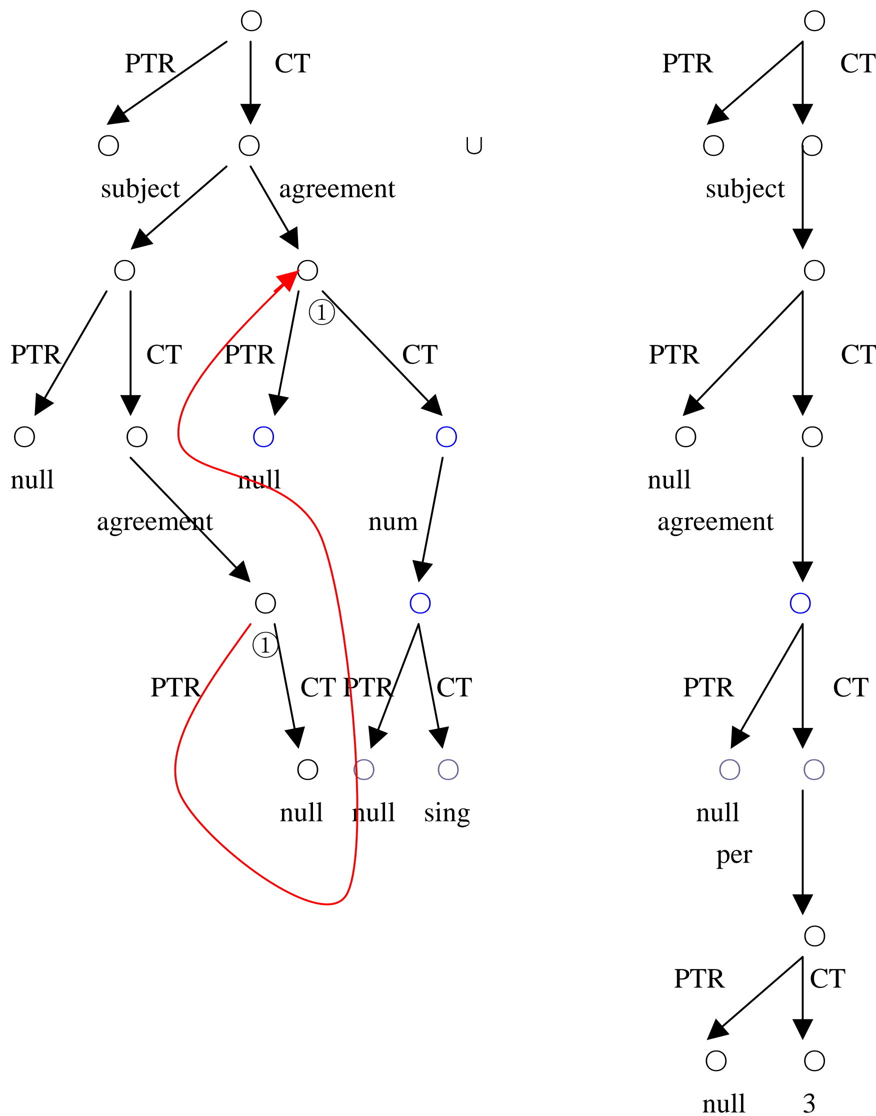
pointer field of f2 is set to f1 completing the unification of the structures and f1 is returned as the value of the unification.

An example: Unify following feature structure.

$$\begin{bmatrix} \text{agreement} & ① \begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agreement} & ① \end{bmatrix} \end{bmatrix}$$

$$\cup \begin{bmatrix} \text{subject} & \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The extended DAGs f1 and f2:



These original arguments are neither identical, nor atomic, nor null, so the main loop is entered. Looping over the features of f2, the algorithm is led to a recursive attempt to unify the values of the corresponding "subject" feature of f1 and f2.

$$\begin{bmatrix} \text{agreement} & ① \end{bmatrix} \cup \begin{bmatrix} \text{agreement} & \begin{bmatrix} \text{per} & 3 \end{bmatrix} \end{bmatrix}$$

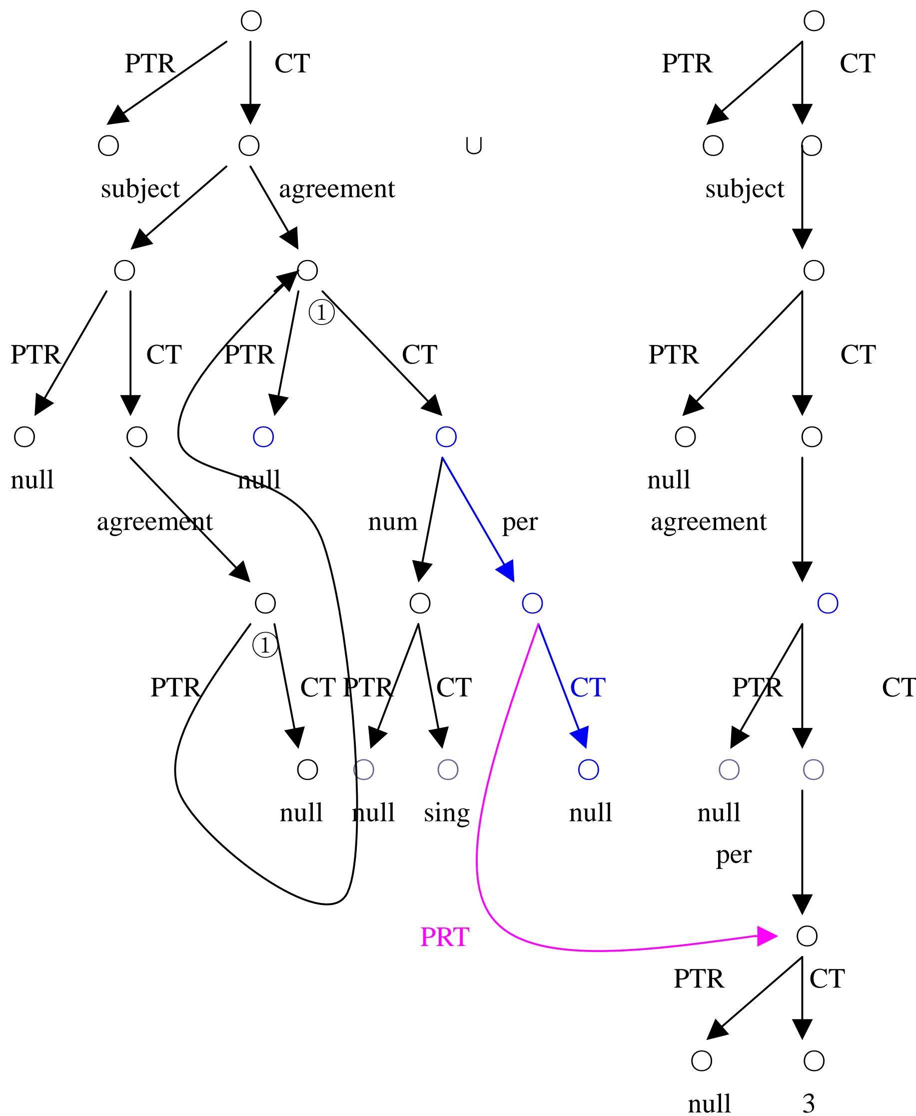
These argument are also non-identical, non-atomic and non-null so the loop is entered again leading to a recursive check of the values of the "agreement" features.

$$\begin{bmatrix} \text{num} & \text{sing} \end{bmatrix} \cup \begin{bmatrix} \text{per} & 3 \end{bmatrix}$$

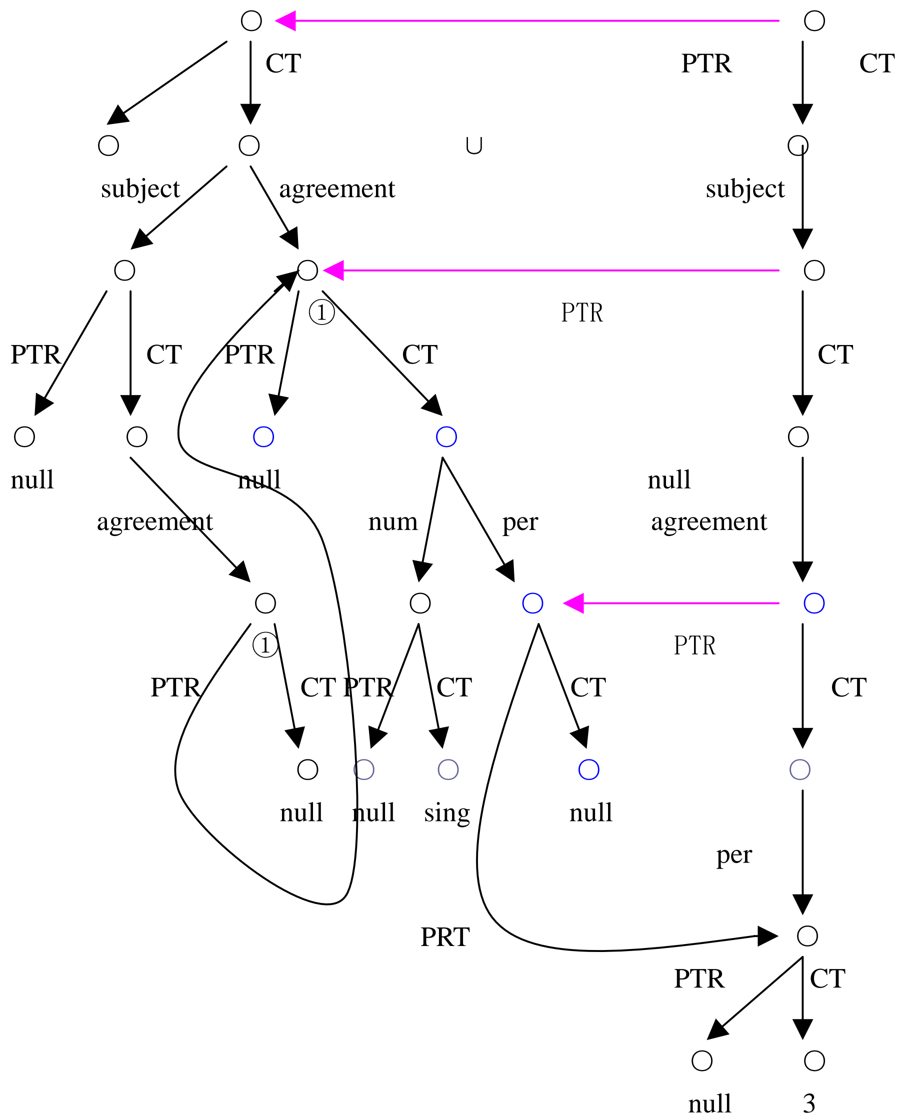
In looping over the features of the second argument, the fact that the first argument lacks "per" feature is discovered. A "per" feature initialized with a "null" value is added to the first argument. This changes the previous unification to the following:

$$\begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & \text{null} \end{bmatrix} \cup \begin{bmatrix} \text{per} & 3 \end{bmatrix}$$

After adding this new "per" feature, the next recursive call leads to the unification of the "null" value of the new feature in the first argument with the 3 value of the second argument. This recursive call results in the assignment of the pointer field of the first argument to the 3 value in f2.



Since there are no further features to check in the f2 argument at any level of recursion. Each in turn sets the pointer for its f2 argument to point at its f1 argument and returns it. The result of all arguments is as following:



4.3.3 Parsing with unification constraints

The CFG rule with unification constraint is as follows:

S → NP VP

<NP head agreement> = <VP head agreement>

<S head> = <VP head>

Its AVM is:

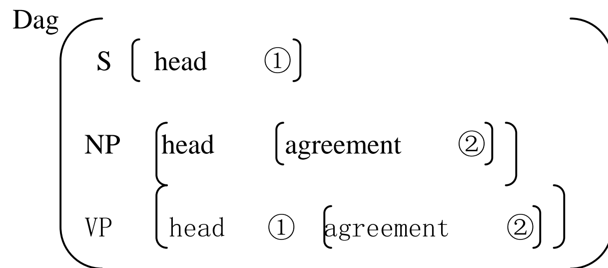
$$\begin{bmatrix} S & \begin{bmatrix} head & ① \end{bmatrix} \\ NP & \begin{bmatrix} head & \begin{bmatrix} agreement & ② \end{bmatrix} \end{bmatrix} \\ VP & \begin{bmatrix} head & ① & \begin{bmatrix} agreement & ② \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

This AVM can be represented by a DAG. So we can use AVM to represent the DAG.

In Earley parser, we can add the DAG to the rule:

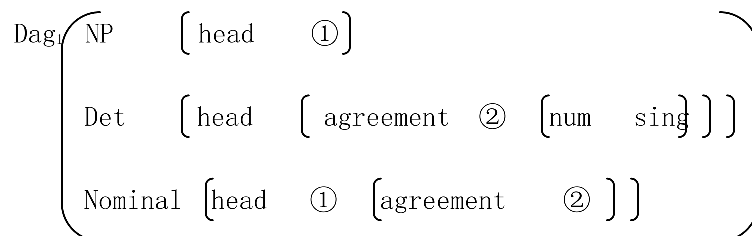
S → . NP VP, [0, 0], [ ], Dag

[ ] means that the parsing just starts. It marks the position of dot of rule in the DAG.



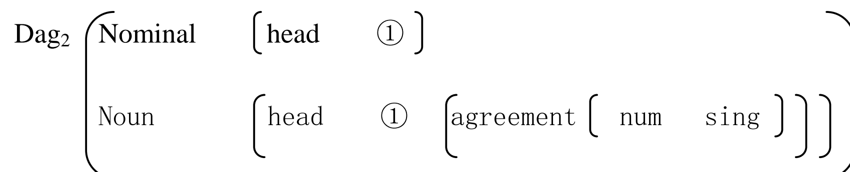
In the chart, it is an active edge.

NP → Det. Nominal, [0, 1], [S_{det}], Dag_1



It is also an active edge.

Nominal → Noun., [1,2], [S_{noun}], Dag_2



It is an inactive edge.

By his means, we can integrate unification into Earley parser.

4.4 Types and Inheritance

The basic feature structures have two problems that have led to extensions to the formalism:

- First problem: there is no way to place a constraint on what can be the value of a feature.

For example, in our current system, there is nothing to stop "num" from have the value 3rd or feminine as values:

[num feminine]

This problem has caused many unification-based grammatical theories to add various mechanisms to try constrain the possible values of a feature. E.g.

FUG (Functional Unification Grammar, Kay, 1979), LFG (Lexical Functional Grammar, Bresnan, 1982), GPSG (Generalized Phrase Structure Grammar, Gazdar et al\., 1985), HPSG (Head-Driven

Phrase Structure Grammar, Pollard et al., 1994).

- Second problem: In the feature structure, there is no way to capture generalization across them. For example, the many types of English verb phrases share many features, as do the many kinds of sub-categorization frames for verbs.

A general solution to both of these problems is the use of types.

Type system for unification grammar has the following characteristics:

- Each feature structure is labeled by a type.
- Each type has appropriateness conditions expressing which features are appropriate for it.
- The types are organized into a type hierarchy, in which more specific types inherit properties of more abstract one.
- The unification operation is modified to unify the types of feature structures in addition to unifying the attributes and values.

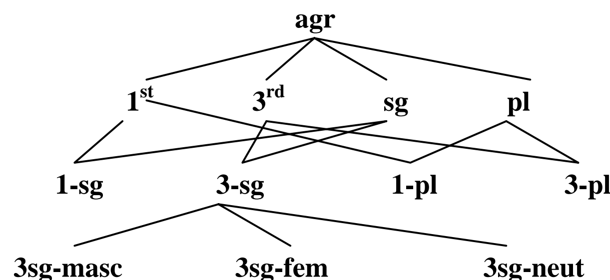
In such typed feature structure systems, types are a new class of objects, just like attributes and values for standard feature structures.

There are two kinds of types:

1. Simple types (atomic types): It is an atomic symbol like **sg** or **pl,** and replaces the simple atomic values used in standard feature structures.

   All types are organized into a multiple-inheritance type hierarchy (a partial order or lattice).

   Following is a type hierarchy of new type **agr**, which will be the type of the kind of atomic object that can be the value of an AGREEMENT feature.



   In this hierarchy, **3rd** is a subtype of **agr**, and **3-sg** is a subtype of both **3rd** and **sg**.

   The unification of any two types is more specific type than the two input types. Thus

   $\mathbf{3^{rd}} \cup \mathbf{sg} = \mathbf{3sg}$

   $\mathbf{1^{st}} \cup \mathbf{pl} = \mathbf{1pl}$

   $\mathbf{1^{st}} \cup \mathbf{arg} = \mathbf{1^{st}}$

   $\mathbf{3^{rd}} \cup \mathbf{1^{st}} = \perp$ (undefined, fail type)

2. Complex types: The complex types specify:

- A set of features that are appropriate for that type.
- Restrictions on the values of those features (expressed in terms of types).
- Equality constraints between the values.

For example, the complex type **verb** represents agreement and verb morphological form information.

A definition of verb would define two appropriate features:

- AGREE: It takes values of type **arg** defined above. :
- VFORM: It takes values of type **vform** which subsumes the seven subtypes: **finite, infinitive, gerund, base, present-participle, past-participle, passive-participle.**

Thus **verb** would be defined as follows:

$$\begin{bmatrix} \textbf{verb} & \\ \text{AGREE} & \textbf{arg} \\ \text{VFORM} & \textbf{vform} \end{bmatrix}$$

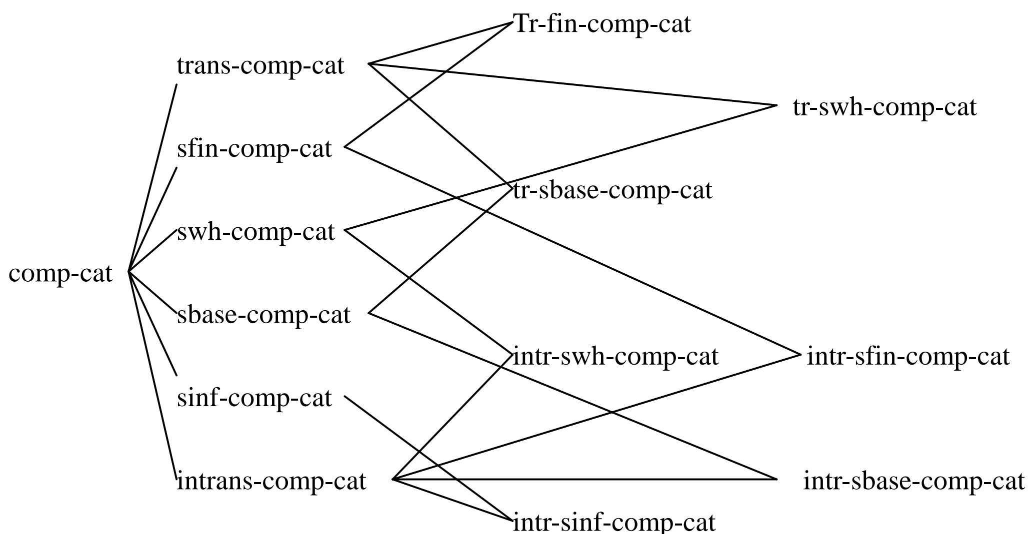
The type noun might be defined with the AGREE feature, but without the VFORM feature.

$$\begin{bmatrix} \textbf{noun} & \\ \text{AGREE} & \textbf{arg} \end{bmatrix}$$

The unification of typed feature structures:

$$\begin{bmatrix} \textbf{verb} & \\ \text{SGREE} & \mathbf{1^{st}} \\ \text{VFORM} & \textbf{gerund} \end{bmatrix} \cup \begin{bmatrix} \textbf{verb} & \\ \text{AGREE} & \textbf{sg} \\ \text{VFORM} & \textbf{gerund} \end{bmatrix} = \begin{bmatrix} \textbf{verb} & \\ \text{AGREE} & \textbf{1-sg} \\ \text{VFORM} & \textbf{gerund} \end{bmatrix}$$

Complex types are also part of the type hierarchy. Subtypes of complex types inherit all the feature of their parents, together with the constraints on the values. Following is a small part of this hierarchy for the sentential complement of verb (Sanfilippo, 1993):



Ex:

tr-swh-comp-cat: "Ask yourself whether you have become better informed."

intr-swh-comp-cat: Mosieur asked whether I wanted to ride."

It is possible to represent the whole phrase structure rule as a type. Sag and Wasow (1999) take a type **phrase** which has a feature called DTRS (daughters), whose value is a list of **phrases**. The phrase "I love Seoul" could have the following representation (showing only the daughter feature):

$$\begin{bmatrix} \textbf{phrase} & \\ \text{DTRS} & \begin{bmatrix} \text{CAT PRO} \\ \text{ORTH} \quad \text{I} \end{bmatrix}, \begin{bmatrix} \text{CAT} \quad \text{VP} & \\ \text{DRTS} & \begin{bmatrix} \text{CAT} \quad \text{V} \\ \text{ORTH LOVE} \end{bmatrix}, \begin{bmatrix} \text{CAT} \quad \text{NP} \\ \text{ORTH} \quad \text{SEOUL} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The resulting typed feature structures place constraints on which type of values a given feature can take, and can also be organized into a type hierarchy. In this case, the feature structures can be well typed.