ISO TC 37/SC 4 N033 Rev. 3
2004-01-02

Language Resource Management

Descriptors and Mechanisms for Language Resources

File ID SC4N033.doc (479ko)
SC4N033.pdf (286 ko)

Title:	Draft - Language Resource Management - Feature Structures - Part I: Feature Structure Representation
Editor(s):	Kiyong Lee
Source:	WG1
Project number:	24610-1 - This reference will supersede all previous one and remain attached as working ref along the duration of the project.
Status:	pre-DIS: reviewed by the Korean group of experts
Date:	2004-01-02
Agenda/Action:	For review by TEI-ISO joint group
References:	WG1 N17; WG1 N23; TEI

Mr. Key-Sun Choi - SC4 Secretary - KORTERM - 373-1
Guseong-dong Yuseong-gu - Daejeon 305-701 - Korea
Tel: +82 42 869 35 25 - Fax: +82 42 869 87 90 - kschoi@cs.kaist.ac.kr -
http://tc37sc4.org

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Copyright notice

Table of Contents

Foreword

(to be filled in)

Introduction

This standard proposal results from the agreement between the Text Encoding Initiative Consortium and the ISO committee TC 37/SC 4 that a joint activity should take place to revise the two existing chapters on Feature Structures and Feature Structure Declaration in the TEI guidelines. This work should lead to both a thorough revision of the guidelines and the production of an ISO standard on Feature Structure Representation and Declaration.

This standard is organized in two separate main parts. The first part is dedicated to the description of what feature structures are, providing an informal and yet explicit outline of their basic characteristics, as well as an XML-based structured way of representing feature structures. This preliminary task is designed to lay a basis for constructing an XML-based reference format for exchanging feature structures between applications. The second part aims at providing an implementation standard for XML-based feature structures, first by formulating constraints on a set of features and a set of their appropriate values and then by introducing a set of wellformedness conditions on feature structures for particular applications, especially related to the goal of language resource management.

1 Scope

Feature structures are an essential part of many linguistic formalisms as well as an underlying mechanism for representing the information consumed or produced by and for language engineering components. This international standard provides a format to represent, store or exchange feature structures in natural language applications, both for the purpose of annotation or production of linguistic data. It also provides a computer format to describe the constraints that bear on a set of features, feature values, feature specifications and operations on feature structures, thus offering means to check the conformance of each feature structure with regards to a reference specification.

2 Normative References

To be checked by LR

ISO/IEC 639, Information technology – ISO 639: 1988, Code for the representation of names of languages.

ISO 639-2: 1998, Code for the representation of names and languages – part 2: Alpha-3 code.

ISO/IEC 646: 1991, Information technology – ISO 7-bit coded character set for information interchange.

ISO 3166-1: 1997, Code for the representation of names of countries and their subdivisions – Part 1: Country codes

ISO 8601: 1988, Data elements and interchange formats - Information interchange – Representation of dates and times.

ISO 8879: 1986 (SGML) as extended by TC2 (ISO/IEC JTC 1/SC 34 N 029: 1998-12-06) to allow for XML.

ISO/IEC 10646-1: 2000, Information technology - Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and basic multilingual plane.

ISO 12620, Computer applications in terminology – Data categories.

Erjavec: Lou complains that the TEI is not in the bibliography. But shouldn't it actually be in this section, as the normative part frequently refers to it?
Also, why is ISO 3166 (names of countries) needed?

3 Terms and Definitions

KATS: The current list of terms listed here is restricted to those in Section 4. It should be augmented with key terms in other sections, especially Section 5.

3.1 atomic value

In a feature structure, the value of each feature is either atomic or complex. An atomic value is some primitive type of object without internal structure. See **complex value**.

3.2 attribute-value matrix

AVM
a very common notation in a matrix form which represents a feature structure consisting of pairs of an attribute, namely feature, and its value
Note: The acronym AVM stands for "Attribute-Value Matrix" where each row represents a pair of a feature and its value, separated by a colon (:), space ( ) or the equality sign (=).

3.3 boxed integer

integer in a box like $\boxed{1}$ which is used for marking structure sharing in an AVM
Note: The index need not be an integer, but can be any alpha-numeric symbol that can be used as a coreferential index.

3.4 compatibility

Two feature structures are compatible if and only if none of the features that they have in common has a conflicting value. On the other hand, two incompatible feature structures contain at least one identical feature which has a conflicting value.

3.5 complex value

The value of a feature in a feature structure can be complex. The complex value is normally a feature structure, thus making a feature structure contain another feature structure within itself in a recursive manner. It can, however, be a list, a set or a bag for some extensions.

3.6 directed acyclic graph

DAG
graph on which each node, except for the terminal ones, points to other nodes or at least one other node, but it disallows any path or any of its segments that points to itself
Note: A feature structure is often represented by a DAG.

3.7 empty path, the

path corresponding to the root node of a graph which represents the empty feature structure without any feature specification
Note: It is represented as a single dot, possibly labelled with the name of a type, on a graph. It may also be represented as [ ] or [$^{\tau}$] labelled with the name of a type τ in the AVM notation.

3.8 feature

attribute or property of an object being described
Note: By taking an appropriate value for the described object, it constitutes part of a feature structure.

3.9 feature specification

assignment of a particular value to a feature in a feature structure

3.10 feature structure

a set of feature specifications which carry partial information about some object being described by assigning a value to each of its features
Note: The feature structure is defined in set-theoretic terms as a partial function from features to values.

3.11 identity element

The empty feature structure is an identity element of the operation called *unification* on feature structures, since it yields the identical result when unified with any other feature structure just as the number 0 is an identity element for the algebraic operation called *addition* on natural numbers.

3.12 graph notation

A single-rooted, labelled and directed graph is often used to represent a feature structure. Each graph representing a feature structure starts with a particular single node called *the root*. From the root, more than one arcs, each of which is labelled with the name of a feature, may branch out to other nodes. These nodes may each terminate with an atomic value or some of them may again branch out to other nodes. For a typed feature structure, each node including the root is labelled with the name of a type.

3.13 multiple inheritance

3.14 path

sequence of feature names which label each of the arcs in a descending order from the root on the graph notation
The notion of *path* can also be extended in the same manner to other notations.

3.15 reentrancy

structure sharing in a feature structure
It may be represented in the graph notation as two or more paths pointing to the same node. These paths are then called *equivalent*, having the same value. As a result, these two or more paths leading to that intersecting node share their values. In the AVM notation, reentrancy is conventionally marked by a boxed integer or alphabetic symbol like $\boxed{3}$ by tagging it to the left of the feature structure or the type name of that node and also at the place of the value being shared by the other paths without copying the shared feature structure or feature value. See **shared value**.

3.16 root, the

topmost node on a graph or an (upside-down) tree that has no ancestors

3.17 shared value

feature value shared by two or more features in a feature structure
In graph notation, a node to which two or more paths merge represents the value shared by the paths. In matrix notation, the shared value is represented by an identical boxed alpha-numeric index. See **reentrancy**.

3.18 subsumption

a reflexive, symmetric and transitive relation between two feature structures
A feature structure A is said to subsume a feature structure B, formally represented as $A \sqsubseteq B$, if A is not more informative than B, or A contains a subset of the feature specifications in B.

3.19 type

Elements of any domain can be sorted into some classes in a structured way, based on similarities of properties. These classes are called “types”.

Note: In linguistics, for instance, class names like *phrase*, *word*, *pos* (parts of speech), *noun*, and *verb* are often taken as types.

3.20 typed feature structure

feature structure that is labelled by the name of a type
In the graph notation, each node on a graph is labelled with the name of a type.

3.21 type inheritance hierarchy

Types are ordered in some hierarchical order so that objects of a lower type inherit properties of their super-types. In linguistics, these hierarchies are often used to organize linguistic descriptions, especially lexical information.

3.22 unification

a binary operation on feature structures that combine two compatible feature structures into one representing exactly all the information contained in the feature structures being unified

4 General Characteristics of Feature Structure

4.1 Overview

A *feature structure* is a general-purpose data structure that identifies and groups together individual *features* by assigning a particular value to each of them. Because of the generality of feature structures, they can be used to represent many different kinds of information. Interrelations among various pieces of information and their instantiation in markup provide a *metalanguage* for representing linguistic content analysis and interpretation. Moreover, this instantiation allows a specification of a set of features with values to be of specific *types*, and also a set of restrictions to be placed on the values for particular features, by means of *feature system declarations*, which are properly discussed in the second part of this standard. Such restrictions provide the basis for at least partial validation of the feature-structure encodings that are used.[1]

4.2 Use of Feature Structures

Feature structures may be understood as providing *partial information* about some object which is described by specifying *values* of some of its features. Suppose we are describing a female employee named Sandy Jones who is 30 years old. We can then talk about at least that person's sex, name and age in a succinct manner by assigning a value to each of these three features of hers. These pieces of information can be put into a simple set notation, as in:

(1) About an employee
{<SEX, *female*>, <NAME, *Sandy Jones*>, <AGE, *30*>}

[1] See illustration examples in non-normative Annex A.

The use of feature structures can easily be extended to linguistic descriptions, too. The phoneme /p/ in English, for instance, can be analyzed as a complex of its distinctive features. It can be partially described as a consonantal, anterior, voiceless, non-continuant or stop sound segment. By introducing the boolean values *plus*(+) and *minus*(-), these features can then be listed as a set consisting of pairs of a feature and its value:

(2) Distinctive features of the sound segment /p/
{<CONSONANTAL, + >, <ANTERIOR, + >, <VOICED, - >, <CONTINUANT, - >}

As a result, it can be distinguished from other phonemes exactly in what aspect they are different from each other. It differs from the phoneme /b/ in VOICING, while it differs from the phoneme /k/ in their articulatory positions: one is articulated at the ANTERIOR, namely lips of the mouth and the other at the non-anterior part, namely back of the oral cavity.

This feature analysis can be extended to the description of other linguistic entities or structures. Consider the verb like 'love'. Its features can be divided into syntactic and semantic properties: as a transitive verb, it takes an object as well as a subject as its arguments, expressing the semantic relation of loving between two persons or animate beings. The exact representation of these feature specifications requires a detailed elaboration of what feature structures are. For now, we can roughly represent these grammatical features in a set format like the following:

(3) Grammatical features of the verb 'love'
{<POS, *verb*>, <VALENCE, *transitive*>, <SEMANTIC_RELATION, *loving*>}

Since its first extensive use in generative phonology in mid-60's, a feature structure has become an essential tool not only for phonology, but also for doing syntax and semantics as well as building lexicons, especially related to computational work. Feature structures are used to describe and model linguistic entities and phenomena by analyzing them as complexes of their properties. For this purpose, it is considered necessary to specify some of the formal properties of feature structures and ways of representing them in a systematic manner.

4.3 Basic Concepts

Feature structures may be viewed in a variety of ways. The most common and perhaps the most intuitive way is to view them in one of the following ways:

- (i) sets of *feature specifications* that consist of pairs of *features* and their *values*

- (ii) *labelled directed graphs with a single root* where each arc is labelled with the name of a feature and directed to its value.

In set-theoretic terms, a feature structure $\mathcal{FS}$ can thus be defined as a *partial function* from a set **Feat** of features to a set **FeatVal** of values, where **FeatVal** consists of a set **AtomVal** of atomic values and a set **FS** of feature structures.

(4) Feature structure as a set or partial function

$\mathcal{FS} \subseteq \{< F_i, v_i > | F_i \in \mathbf{Feat}, v_i \in \mathbf{FeatVal}\}$
or
$\mathcal{FS} : \mathbf{Feat} \longrightarrow \mathbf{FeatVal}$,
where $\mathbf{FeatVal} = \mathbf{AtomVal} \cup \mathbf{FS}$.

In a feature structure, feature values are either atomic or complex. Atomic values are linguistic entities without internal structure, while complex values are themselves feature structures.

Here is a linguistic example. The part of speech feature, abbreviated as POS, takes the name of an atomic category like *verb* as value. The agreement feature AGR in English, on the other hand, takes a complex value in the form of a feature structure where the features PERSON and NUMBER are appropriately specified. The word 'love', for instance, is here analyzed as having the value of its POS feature specified as *verb* and the value of its AGR feature specified by a feature structure consisting of two feature specifications: one specifies the value of PERSON with *3rd* and the other, the value of NUMBER with *singular*.

4.4 Notations

As has been shown, a feature structure does not simply list feature specifications. It can recursively embed another feature structure into itself because some of the features take featur structures as values. Hence, to represent these complex feature structures in a manner easy to understand, clear and exact notations are necessary.

Just as the notion of feature structures may be conceived in different ways, there are a few different ways of representing them. Here are the three notations that are most commonly used: a graph, a matrix and an XML-based notation. Graphs are for mathematical discourses, matrices for linguistic descriptions, and XML notations for computational implementation.

4.4.1 Graph Notation

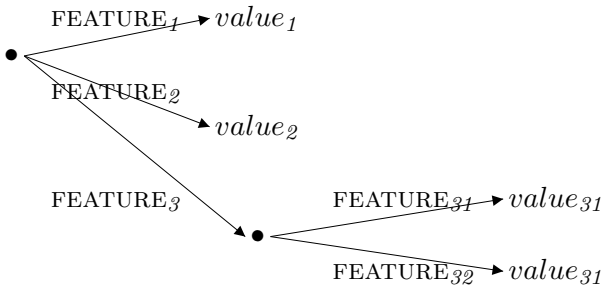
For conceptual coherence and mathematical elegance, feature structures are often represented as labelled directed graphs with a single root.[2]

Each graph starts with a single particular node called *the root*. From this root, any number of *arcs* may branch out to other nodes and then some of them may terminate or extend to other nodes. The extension of directed arcs must, however, stop at some terminal nodes. On such a graph which is representing a feature structure, each arc is labelled with a *feature* name and its directed node, labelled with its *value*.

Here is a very simple example for a directed graph representing a feature structure.

(5) Feature structure in graph notation

[2]This graph can be either (1) *acyclic*, thus allowing the acronym DAG for feature structures or (2)*cyclic* for handling cases like the Liar's paradox.
DAGs are of course by definition not cyclical. And feature structures are usually represented as DAGs; but there have been some suggestions that cyclical feature structures should be introduced to model some linguistic phenomena.



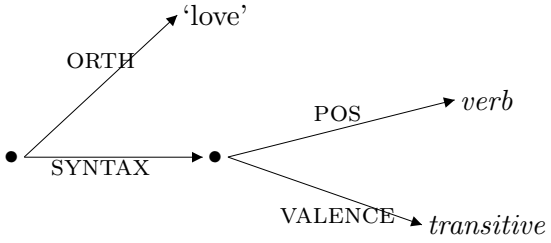
In this graph, the two features FEATURE$_1$ and FEATURE$_2$ are atomic-valued, taking *value*$_1$ and *value*$_2$ on the terminal nodes respectively as their value. The feature FEATURE$_3$ is, however, complex-valued, for it takes as its value the feature structure which is represented by the two arcs FEATURE$_{31}$ and FEATURE$_{32}$ with their respective values, *value*$_{31}$ and *value*$_{32}$.

A graph may just consist of the root node only, that is, without any branching arcs. Such a graph represents the *empty feature structure*.

From the root, more than one arcs may branch out and each of them forms a sequence of feature names of length 1. Here, each sequence consists of a single feature name. Some of these labelled arcs originating from the root may again stretch out to another node and then from this node to another, forming an indefinitely long sequence of feature names. Such a sequence of feature names, labelling the arcs from the unique root node to each of the terminal nodes on a graph, is called *path*. For example, there are four paths in (5): FEATURE$_1$, FEATURE$_2$, FEATURE$_3$.FEATURE$_{31}$ and FEATURE$_3$.FEATURE$_{32}$.

Here is a linguistically more relevant example.

(6) Linguistic example in graph notation



This graph consists of three paths: ORTH, SYNTAX.POS, and SYNTAX.VALENCE. The path consisting of a single feature name ORTH is directed to the terminal node 'love', which is an atomic value. The path SYNTAX.POS terminates with the atomic value *verb* and the path SYNTAX.VALENCE with the atomic value *transitive*. The non-terminal feature SYNTAX takes a complex value, namely a feature structure consisting of the two feature specifications, <POS, *verb*> and <VALENCE, *transitive*>.

4.4.2 Matrix Notation

Despite its mathematical elegance, graphs cause problems of typesetting and readability when they get complex. To remedy some of these problems, feature structures are more often depicted in a matrix notation called *attribute-value matrix*, or simply AVM.[3]

[3]The term 'feature' is sometimes called *attribute*.

(7) Matrix notation

$$\begin{bmatrix} \text{FEATURE}_1 & value_1 \\ \text{FEATURE}_2 & value_2 \\ \text{FEATURE}_3 & \begin{bmatrix} \text{FEATURE}_{31} & value_{31} \\ \text{FEATURE}_{32} & value_{32} \\ \multicolumn{2}{c}{\ldots} \\ \text{FEATURE}_{3k} & value_{3k} \end{bmatrix} \\ \multicolumn{2}{c}{\ldots} \\ \text{FEATURE}_n & value_n \end{bmatrix}$$

Each row in the square bracket with a FEATURE name followed by its *value* name represents a feature specification in a feature structure.[4] Feature values can be either atomic or complex. Each row with an atomic value terminates at that value. But if the value is complex, then that row leads to another feature structure, as in the case of FEATURE_3 above.

The notion of *path* is also important in the AVM notation for its applications that will be discussed presently. A *path* in an AVM is a sequence of feature names, as is the case with feature structure graphs. If an AVM has no row and thus no occurrences of features, being represented as [ ], such an AVM represents the empty feature structure and only has the empty path of length 0. But if an AVM has at least one row consisting of a feature name and its value, then there is a path of length 1 corresponding to each occurrence of a feature name in each row. Given a path of length i, if the last member of that path takes a nonempty feature structure as value, then that path forms a new path of length $i + 1$ by taking each one of the features in that feature structure as its member.

For illustration, consider the following AVM which represents exactly the same feature structure as is represented by the graph notation (6).

(8) Example of an AVM notation

$$\begin{bmatrix} \text{ORTH} & \text{`love'} \\ \text{SYNTAX} & \begin{bmatrix} \text{POS} & verb \\ \text{VALENCE} & transitive \end{bmatrix} \end{bmatrix}$$

This AVM has three paths: ORTH, SYNTAX.POS and SYNTAX.VALENCE. Note that the third path starts with SYNTAX, although it does not occur in the third row.

4.4.3 XML-based Notation

Representing feature structures in XML notation is possible and rather straightforward, although a fuller implementation might involve various problems and possibilities. For example, the feature structure in the following shows a way to represent the same feature structure given in (6) and (8) in the above.

(9) Feature structure in XML notation

[4] A colon, an equality sign or a little empty space separates a feature from its value on each row of an AVM.

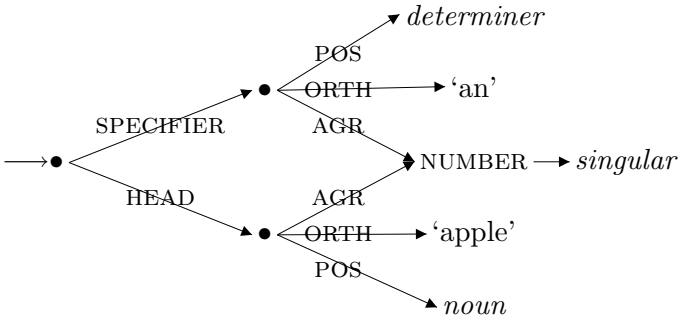
```
<fs>
    <f name="orth"><str>love</str></f>
    <f name="syntax">
        <fs>
        <f name="pos"><sym value="verb"/></f>
        <f name="valence"><sym value="transitive"/></f>
        </fs>
    </f>
</fs>
```

The feature structure in XML notation is surrounded in <fs> tags, and each of its features with <f> tags. Note that despite the apparent difference between the representations (6), (8) and (9), the information contained in each structure corresponds quite systematically with that in other structures. For example, arc labels in graph notation like ORTH and SYNTAX are now represented by <f> tags with appropriate names. A detailed discussion including ways to simplify the representation in (9) will be provided in Section 5.

4.5 Structure Sharing

The graphic notation can clearly represent *structure sharing* also called *reentrancy*. Consider the following:

(10) Merging paths in graph notation



Here, the two SPECIFIER.AGR and HEAD.AGR paths merge on the node NUMBER, indicating that they share one and the same feature structure as their value.

Such structure sharing can also be represented in an AVM.

(11) Structure sharing in AVM notation

$$\begin{bmatrix} \text{SPECIFIER} & \begin{bmatrix} \text{POS } \textit{determiner} \\ \text{ORTH 'an'} \\ \text{AGR } \boxed{1} \left[\text{NUMBER } \textit{singular}\right] \end{bmatrix} \\ \text{HEAD} & \begin{bmatrix} \text{POS } \textit{noun} \\ \text{ORTH 'apple'} \\ \text{AGR } \boxed{1} \end{bmatrix} \end{bmatrix}$$

The two occurrences of the identical boxed integer, namely $\boxed{1}$, in the above representation show that those two occurrences of the feature AGR share the same value, namely *singular*, for the feature NUMBER.

The XML-based notation can also handle such sharing or reentrancy by introducing a global attribute `n` on element `f`.

(12) Structure sharing in XML notation

```
<fs>
  <f name="specifier">
     <fs>
        <f name="agr" n="@1">
           <fs>
             <f name="number"><sym value="singular"/></f>
           </fs>
        </f>
        <f name="pos"><sym value="determiner"/></f>
     </fs>
  </f>
  <f name="head">
     <fs>
       <f name="agr" n="@1"/>
       <f name="pos"><sym value="noun"/></f>
     </fs>
  </f>
</fs>
```

Reentrancy is symmetric and there is no distinguished representative among the different occurrences of a shared node. In particular, this implies that a value may be attached to all occurrences of a shared label:

(13) Specifying shared values

```
<fs>
  <f name="specifier">
     <fs>
        <f name="agr" n="@1">
           <fs>
             <f name="number"><sym value="singular"/></f>
           </fs>
        </f>
        <f name="pos"><sym value="determiner"/></f>
     </fs>
  </f>
  <f name="head">
     <fs>
       <f name="agr" n="@1">
          <fs>
             <f name="number"><sym value="singular"/></f>
          </fs>
       </f>
       <f name="pos"><sym value="noun"/></f>
```

```
      </fs>
    </f>
  </fs>
```

Related to structure sharing are at least the following two issues: underspecified values and cyclic feature structures.

4.5.1 Sharing Underspecified Values

Two nodes of a feature structure may be shared, even if their value is underspecified:

(14) Sharing of underspecified values

$$\begin{bmatrix} \text{SPECIFIER} & \begin{bmatrix} \text{AGR} & \boxed{1} \\ \text{POS} & \textit{determiner} \end{bmatrix} \\ \text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1} \\ \text{POS} & \textit{noun} \end{bmatrix} \end{bmatrix}$$

In that case, as will be discussed in Seciton 5 more in detail, the element **any** may be used to denote underspecified values for a feature for XML notation:

(15) Use of the element **any** for underspecified values

```
<fs>
  <f name="specifier">
     <fs>
        <f name="agr" n="@1"><any/></f>
        <f name="pos"><sym value="determiner"/></f>
     </fs>
  </f>
  <f name="head">
     <fs>
       <f name="agr" n="@1"/>
       <f name="pos"><sym value="noun"/></f>
     </fs>
  </f>
</fs>
```

The atomic value **any** can be literally any value and its dual is **none**.[5]

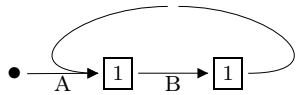
4.5.2 Cyclic Feature Structures

The current proposal does not forbid the representation of cyclic feature structures, even if not handled by most FS implementations.

The most straightforward case is provided by direct embedding:

[5] See Shieber (1986: 43-44) for a fuller discussion.

(16) Direct embedding in AVM



(17) Direct embedding in AVM

$$\left[\text{A } \boxed{1} \left[\text{B } \boxed{1}\right]\right]$$

(18) Direct embedding in XML

```
<f name="a" n="@1">
   <fs>
     <f name="b" n="@1"/>
   </fs>
</f>
```

However, a cycle may be indirect as follows:

(19) Indirect cycle in AVM

$$\left[\begin{array}{l}\text{A } \boxed{1} \left[\text{B } \boxed{2}\right] \\ \text{C } \boxed{2} \left[\text{D } \boxed{1}\right]\end{array}\right]$$

(20) Indirect cycle in XML

```
<fs>
  <f name="a" n="@1">
     <fs>
       <f name="b" n="@2"/>
     </fs>
  </f>
  <f name="c" n="@2">
     <fs>
       <f name="d" n="@1"/>
     </fs>
  </f>
</fs>
```

These equivalent examples show that the two structures tagged by $\boxed{1}$ and $\boxed{2}$ share their values with each other in cycle.

4.6 Multi-valued Features

In feature-based grammar formalisms, such as HPSG and LFG, multi-valued features are very common. Some features take list values and others set or multiset values.[6] All the elements in a list are ordered, while there is no such ordering in a set or multiset. In this sense, multisets are still sets. Nevertheless, multisets differ from ordinary sets in that multisets

[6]The term 'multiset', with its notation {. .} or $\{\}_m$, is often called 'bag'.

allow the occurrence of identical elements in them as lists do. The multiset {. a, a .}, for instance, is not the same as {. a .}, but the ordinary sets {a, a} and {a} are the same because of the principle of extensionality.

4.6.1 List-valued Features

Perhaps the most famous example of a list-valued feature is the SUBCAT feature in HPSG. It is used to describe the kind of a grammatical subject and objects that a verb expects ("subcategorizes for"). For example, to represent that the English verb form 'gives' as used in structures like 'John gives Mary a kiss' expects a nominative noun phrase as the subject, an accusative noun phrase as its indirect object, and another accusative noun phrase as the direct object, and expects these elements in this order, the feature SUBCAT is given the value:

(21) SUBCAT as a list-valued feature

$$\left[\text{SUBCAT}\left\langle \text{NP}[nom],\ \text{NP}[acc],\ \text{NP}[acc]\right\rangle\right]$$

Here, each of the NPs may carry a label in the form of a boxed integer to indicate structure sharing with the arguments of a predicate that expresses the semantics of the verb, as shown in figure NN.

In a more recent version of HPSG, the feature SUBCAT is replaced by a new name, VALENCE, that takes as its value a feature structure which consists of two list-valued features SPECIFIER and COMPS, complements.[7] They are then linked to another list-valued feature ARG-STR, argument structure. Here is an example:

(22) List-valued features

$$\left[\begin{array}{ll}\text{ORTH} & \text{'gives'}\\ \text{SYNAX} & \left[\begin{array}{ll}\text{POS} & verb\\ \text{VALENCE} & \left[\begin{array}{ll}\text{SPECIFIER} & \left\langle \boxed{1}\ \text{NP}[3sing]\right\rangle\\ \text{COMPS} & \left\langle \boxed{2}\ \text{NP},\ \boxed{3}\ \text{NP}\right\rangle\end{array}\right]\end{array}\right]\\ \text{ARG-STR} & \left\langle \boxed{1}, \boxed{2}, \boxed{3}\right\rangle\\ \text{SEMANTICS} & \left[\begin{array}{l}\text{RELATION}\left\langle \text{act, giving}\right\rangle\\ \text{GIVER}\ \boxed{1}\\ \text{RECIPIENT}\ \boxed{2}\\ \text{GIVEN}\ \boxed{3}\end{array}\right]\end{array}\right]$$

The order of complements is crucial in English. The indirect object of a ditransitive verb like 'give' or 'buy' precedes the direct object, as in 'John bought Mary a doll'. Hence, the value of the feature COMPS must be presented as a list where each of its members forms an ordered sequence.

[7] The feature SPEC is treated as a list-valued feature because the feature ARG-STR is treated as the $\oplus$ concatenation of the two list values of SPECIFER and COMPS.

Note that a list as a feature value may consist of either atomic or complex values, as shown below.

(23) List as a feature value

$$\begin{bmatrix} \text{F } \langle \text{a, b} \rangle \\ \text{G } \left\langle \left[\text{A a}\right], \left[\text{B b}\right] \right\rangle \end{bmatrix}$$

Here, the feature F takes a list of two atomic objects as its value, whereas the feature G takes a list of two feature structures as its value.

List values can also be represented recursively as shown below:

(24) Recursive representation

$$\begin{bmatrix} \text{F } \begin{bmatrix} \text{FIRST a} \\ \text{REST } \begin{bmatrix} \text{FIRST b} \\ \text{REST null} \end{bmatrix} \end{bmatrix} \\ \text{G } \begin{bmatrix} \text{FIRST } \left[\text{A a}\right] \\ \text{REST } \begin{bmatrix} \text{FIRST } \left[\text{B b}\right] \\ \text{REST null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

According to this representation, lists as feature values must be viewed as simply serving as a shorthand notation for representing recursively built complex feature structures.[8]

4.6.2 Set-valued Features

Some grammatical features may take sets as their values. In free or semi-free order languages like Japanese, Korean or even German, for instance, the subject and the verbal complements in a sentence have no fixed order.[9] Hence, for these languages, the feature COMPS as well as the feature ARG-STR may be treated as taking a set, not a list of complements or arguments as its value. For the German equivalent of 'gives', the verb form 'gibt', we would have the following:

(25) Set-valued features

$$\begin{bmatrix} \text{ORTH 'gibt'} \\ \text{SYNTAX } \begin{bmatrix} \text{POS } \textit{verb} \\ \text{VALENCE } \begin{bmatrix} \text{SPECIFIER } \left\langle \boxed{1}\ \text{NP}[\textit{nom}] \right\rangle \\ \text{COMPS } \left\langle \boxed{2}\ \text{NP}[\textit{dat}], \boxed{3}\ \text{NP}[\textit{acc}] \right\rangle \end{bmatrix} \end{bmatrix} \\ \text{ARG-STR } \left\{ \boxed{1}, \boxed{2}, \boxed{3} \right\} \end{bmatrix}$$

[8] See Shieber (1986: 29-30) for the recursive definition of *list*.

[9] There have been some controversies among practicing linguists concerning the validity of presenting semi-free or free word order as supporting evidence for the introduction of sets as feature values.

The proper description of relative clauses like 'who runs' may require a multi-valued feature RESTRICTION that provides a set of restrictions on some individual PARAMETER.

(26) Set value for the feature RESTRICTION

$$\begin{bmatrix} \text{PARAMETER } \boxed{1} \\ \text{RESTRICTION } \left\{ \begin{bmatrix} \text{RELATION } <\text{property, human}> \\ \text{INSTANCE } \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{RELATION } <\text{activity, running}> \\ \text{RUNNER } \boxed{1} \end{bmatrix} \right\} \end{bmatrix}$$

Here, the set value for the feature RESTRICTION contains two feature structures as its members.[10]

4.6.3 Multiset-valued Features

Some set-valued features are known to take a particular sort of set called multiset or bag as their value. The set-valued feature SLASH in HPSG, for instance, is used for dealing with *wh*-movement and other extraction-like phenomena, where the values of SLASH contains the (properties of the) extracted gaps and these gaps can at times be identical.

For illustration, consider an example of so-called filler-gap constructions.[11]

(27) Filler-gap construction
{This bus}$_1$, I don't think {Palo Alto}$_2$ is very easy to get to t_2 on t_1?

It has been claimed that the gaps or traces t_1 and t_2 in this example require the feature specification like [SLASH {. NP, NP .}].

Here is a more convincing example:[12]

(28) Coreferential pronouns
John$_{1=2}$ is an idiot. But he$_1$ thinks he$_2$ is smart.

The two occurrences of the pronouns above should be coreferential. So to be able to capture this coreferentiality, a multiset like the following must be set up for some feature specification.

(29) Coreferential multiset

$$\{. \begin{bmatrix} \text{POS } \textit{pronoun} \\ \text{PERSON } \textit{3rd} \\ \text{NUMBER } \textit{sing} \\ \text{GENDER } \textit{masc} \end{bmatrix}, \begin{bmatrix} \text{POS } \textit{pronoun} \\ \text{PERSON } \textit{3rd} \\ \text{NUMBER } \textit{sing} \\ \text{GENDER } \textit{masc} \end{bmatrix} .\}$$

4.7 Typed Feature Structure

In many of the recent grammar formalisms, *typed feature structure* has become the main tool for their linguistic descriptions and implementation.

[10] In Pollard and Sag (1994), so-called nonlocal features like QUEstion, RELative and SLASH are treated as taking set values for analyzing so-called filler-gap or unbounded dependency constructions. The feature QUE takes as value a set of *non-pronominals*, the feature REL a set of *referential* indices, and the feature SLASH a set of *local* structures.

[11] Taken from Pollard and Moshier (1990: 291).

[12] Again, taken from Pollard and Moshier (1990: 294).

4.7.1 Types

Elements of any domain can be sorted into some classes called *types* in a structured way, based on commonalities of their properties. Linguistic entities, for instance, like *phrase*, *word*, *pos* (parts of speech), *noun*, and *verb* are treated as features in non-typed feature structure. But in typed feature structure they are rather taken as types.

By *typing*, each feature structure is assigned a particular type. Each feature specification with a particular value is then constrained by this typing. A feature structure of the type *noun*, for instance, would not allow a feature like TENSE in it or a specification of its feature CASE with a value of the type *feminine*.[13]

4.7.2 Definition

The extension of non-typed feature structure to typed feature structure is very simple in a set-theoretic framework. The main difference between them is the assignment of types to feature structures. A formal definition of typed feature structure can thus be given as follows:[14]

(30) Formal definition of typed feature structure

Given a finite set of **Features** and a finite set of **Types**, a typed feature structure is a tuple $\mathcal{TFS} = <$ **Nodes**$, r, \theta, \delta >$ such that

i. **Nodes** is a finite set of nodes.

ii. r is a unique member of **Nodes** called *the root*.

iii. θ is a total function that maps **Nodes** to **Types**.

iv. δ is a partial function from **Features**$\times$**Nodes** into **Nodes**.

First, each of the **Nodes** must be rooted at or connected back to the root r. Secondly, there must one and only one root for each feature structure. Thirdly, each of the **Nodes**, including the root r node and terminal nodes, must be assigned a type by the typing function θ. Finally, each of the **Features** labelling each of **Nodes** is assigned a unique value by the feature value function δ.[15]

4.7.3 Notations

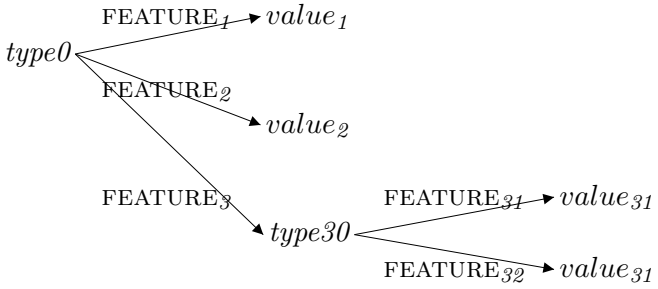
The typing of feature structures can easily be treated in our notations. A graph for a typed feature structure will have the following form:

[13] Note that atomic feature values are considered *types*, too.

[14] Slightly modified from Carpenter (1992: 36).

[15] The unique-value restriction on features does not exclude multi-values or alternative values because even in these cases each feature ultimately takes a single value which may be considered complex in structure.

(31) Typed feature structure in graph



The only difference between the typed graph (31) and the non-typed graph (31) is that each of the two nodes has been assigned a type: one is of *type0* and the other of *type30*.

Corresponding to the non-typed AVM (8), here is a typed AVM:

(32) Typed feature structure in AVM

$$\begin{bmatrix} \textit{word} \\ \text{ORTH } \text{`love'} \\ \text{SYNTAX} \begin{bmatrix} \textit{verb} \\ \text{VALENCE } \textit{transitive} \end{bmatrix} \end{bmatrix}$$

Here, the entire AVM is assigned the type *word*, whereas the inner AVM is assigned the type *verb*. Unlike the non-typed (8), this typed AVM carries an additonal piece of information tells that the features ORTH and SYNTAX are of the type *word* and the feature VALENCE of the type *verb*.

The same type of information can be encoded in an XML notation, as below:

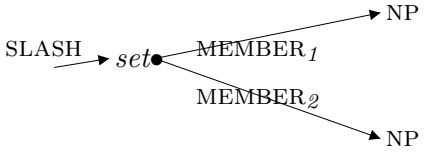
(33) Typed feature structure in XML notation

```
<fs type="word">
    <f name="orth"><str>love</str></f>
    <f name="syntax">
        <fs type="verb">
            <f name="valence"><sym value="transitive"/></f>
        </fs>
    </f>
</fs>
```

Note here that the line <f name="pos"><sym value="verb"/></f> in the embedded feature structure <fs> has been replaced by typing that <fs> as in <fs type="verb">.

The use of *type* may also increase the expressive power of a graph notation. On the typed graph notation, for instance, multi- values can be represented as terminating nodes branching out of the node labelled with the type *set*, *multiset* or *list*. This node in turn is a terminating node of the arc labelled with a multit-valued feature, say SLASH. Each arc branching out of the multi-valued node, say *set*, is then labelled with a feature appropriate to the type.

(34) Set-valued feature SLASH in graph notation



The features like MEMBER$_1$ and MEMBER$_2$ here should be considered appropriate for the type *set*.

4.8 Type Inheritance Hierarchies

Types organize feature structures into natural classes and perform the same role as concepts in terminological knowledge representation systems or abstract data-types in object oriented programming languages. It is therefore natural to think of types as being organized into an inheritance hierarchy based on their generality.

The type inheritance hierarchy is defined by assuming a finite set BF TYPE of types, ordered according to their specificity, where type τ is more specific than type σ if τ inherits all properties and characteristics from σ. In this case σ *subsumes* or is more *general* than τ: for $\sigma, \tau \in \mathbf{Type}, \sigma \sqsubseteq \tau$. If $\sigma \sqsubseteq \tau$, then σ is also said to be a *supertype* of τ, or, inversely, τ is a *subtype* of σ.

The standard approach in knowledge representation systems, which is adopted in the definition of type hierarchies, has been to define a finite number of ISA arcs which link subtypes to supertypes. The full subsumption relation is then defined as the *transitive* and *reflexive* closure of the relation determined by the ISA links. A standard restriction on the ISA links is that they must not be cyclic, i.e. it should not be possible to follow the links from a type back to itself. This restriction makes the subsumption relation a *partial order*.

4.8.1 Definition

An inheritance hierarchy is then formally defined as follows:

(35) Definition of inheritance hierarchy
Given a finite set **Type** of types and the subsumption relation $\sqsubseteq$ over **Type**, an *inheritance hierarchy* is a finite bounded complete partial order $<\mathbf{Type}, \sqsubseteq>$.

This definition simply says that a type hierarchy forms a tree-like finite structure. It must have the following properties:

(36) Properties of type hierarchy

Unique top: It must be a single hierarchy containing all the types with a unique top type.

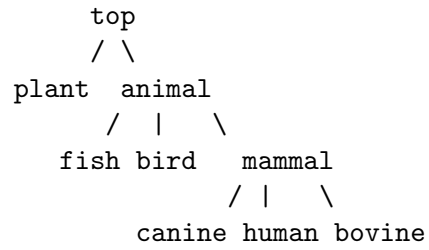
No cycle: It must have no cycles.

Unique greatest lower bounds: Any two compatible types must have a unique highest common descendant or subtype called *greatest lower bound*.[16] Incompatible types share no common descendants or subtypes.

[16] If the most general type is depicted not as the top, but as the bottom such that the hierarchical tree branches out upward like a real tree with the root at the bottom, then this property must be restated as: Any two compatible types must have a unique least upper bound.

Here is an example of a type hierarchy depicting a part of the natural world:

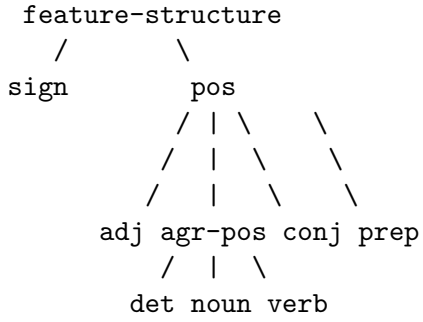
(37) Type hierarchy for some animals



Here, while the types *human* and *canine* are not compatible, the types *animal* and *human* are compatible and thus must have a unique greatest lower bound. Being in the hierarchical relation, the type *human* becomes that lower bound in a trivial manner.

A linguistically more relevant example can be given as below: [17]

(38) Lingusitic example for type hierarchy

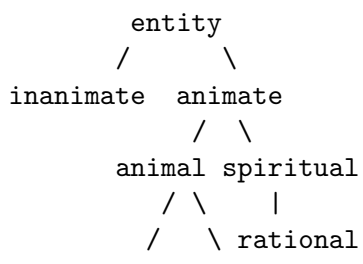


Here the type *feature-structure* is treated as the unique top type. The types *det*, *noun* and *verb* are treated as subtypes of the type *agr-pos*, since they are governed by agreement rules in English.[18]

4.8.2 Multiple Inheritance

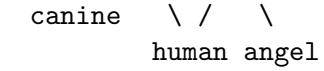
Unlike phrase trees, type hierarchies allow common parents or supertypes. Consider a naive medieval picture of entities as depicted as below:

(39) Medieval hierarchy of entities



[17] Taken modified from Sag, Wasow and Bender (2003: 61).

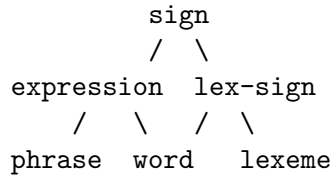
[18] In a language like French or Latin, the type *adj* should be also be treated as a subtype of *agr-pos*.



Subtypes inherit all the properties from their supertypes. The type *human*, for instance, inherits all the properties of its supertypes, both *animal* and *rational*, *spiritual*, *animate* and the top type *entity*. Note that it has two immediate supertypes or *parents*, thus being entitled for so-called *multiple inheritance*. A *human* thus is a *spiritual* and *rational animal* animate being.

Linguistic signs may also allow multiple inheritance like the following.[19].

(40) Multiple inheritance



Here, the type *word* inherits all the properties from both of its immediate supertypes *expression* and *lex-sign*. Hence, a word is a lexical expression.

4.8.3 Type Constraints

In the feature structures discussed so far there is no notion of *type constraints* or simply *typing*: although the nodes in a feature structure graph were labelled with types, arbitrary labellings with type symbols and features were permissible. What is missing is *appropriateness conditions* which model the distinction between features which are not appropriate for a given type and those whose values are simply unknown.

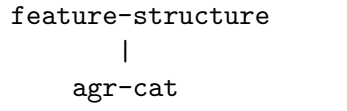
The extension to feature typing is bound to the type hierarchy: for each feature there must be a least type where the feature is introduced and the type of the value for the feature is specified. Furthermore, if a feature is appropriate for a type, then it is appropriate for all of its subtypes.

Consider features like CASE and AUX for English. The feature CASE may be appropriate for the type *noun*, while it may not be so for the type *verb*. Likewise, the feature AUX is appropriate for the type *verb*, but not for the type *nominal*. Hence, each type is closely associated with a set of appropriate features.

The values of each feature are again restricted as types. The appropriate or permissible values of the feature CASE are *nom*, *dat*, *acc* etc., but cannot be boolean values, namely + and -. On the other hand, the appropriate values of the feature AUX are only boolean values.

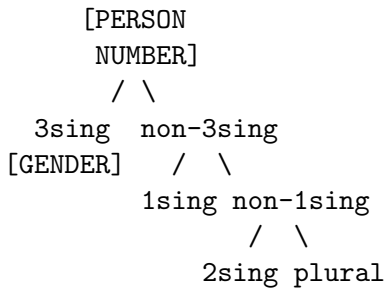
Consider the following type hierarchy for agreement:[20]

(41) Agreement type hierarchy



[19]See Sag, Wasow and Bender (2003: 470-475)

[20]Copied from the type hierarchy presented in Sag, Wasow and Bender (2003: 492).



Here, the type *agr-cat* and its left daughter *3sing* are annotated with a set of *appropriate features*, {PERSON,NUMBER} and {GENDER} for their respective type. By such an annotated hierarchy, the construction of well-formed feature structures is strictly constrained. In constructing feature structures, the type *agr-cat* licenses the specification of the features PERSON and NUMBER only, while the type *3sing* allows the specification of the feature GENDER as well as those two inhertied features *person* and *number* from its supertype *agr-cat*.

Furthermore, each feature is assigned a particular set called *library* of feature values. The following would be an example:

(42) Feature value library

features	admissible values
PERSON	{1st, 2nd, 3rd}
NUMBER	{singular, plural}
GENDER	{feminine, masculine, neuter}

These two working together lay a basis for deciding on *well-formed feature structures*. For example, the following would not be a well-formed feature structure:

(43) Ill-formed feature structure

$$\begin{bmatrix} \textit{agr-cat} \\ \text{PERSON } \textit{3rd} \\ \text{TENSE } \textit{singular} \end{bmatrix}$$

The feature TENSE is not appropriate for the type *agr-cat* nor the value *singular* can be admissible for the feature TENSE. Thus, the feature structure above is declared to be ill-formed.

On the other hand, the following is a well-formed feature structure:

(44) Well-formed feature structure

$$\begin{bmatrix} \textit{1sing} \\ \text{PERSON } \textit{1st} \\ \text{NUMBER } \textit{singular} \end{bmatrix}$$

Being a subtype of *agr-cat*, the type *1sing* inherits two of its appropriate features. These two features are then assigned admissible values.

4.9 Subsumption: Relation on Feature Structures

The primary goal in constructing feature structures is thus to capture and represent partial information. No feature structure is expected to represent the total information describing all possible worlds or states of affairs. It may be of greater interest and value to focus on particular aspects of interesting situations and capture various sorts of related information. The relation of *subsumption* on feature structures is thus introduced to be able to tell which feature structure carries more information than the others.

Some feature structures carry less information than others. The extreme case, perhaps the most uninteresting case, is the *empty* feature structure [ ] sometimes called *variable* that carries no information at all. For more interesting cases, consider the following two feature structures:

(45) a. $\begin{bmatrix} \textit{word} \\ \text{ORTH } \text{`loves'} \end{bmatrix}$

b. $\begin{bmatrix} \textit{word} \\ \text{ORTH } \text{`loves'} \\ \text{SYN } \begin{bmatrix} \textit{verb} \\ \text{FORM } \textit{finite} \end{bmatrix} \end{bmatrix}$

The feature structure (a) says that the word is a string consisting of 5 alphabets spelled as 'l-o-v-e-s' and that's all. But the feature structure (b) says more than that by providing the additional information that it is a finite verb. Hence, (a) is said to be less informative than (b).

To describe such a relation among some feature structures, a technical term is introduced that is called *subsumption*. In the above case, (a) is said to subsume (b).

4.9.1 Definition

Intuitively speaking, a feature structure A subsumes a feature structure B if A is not more informative than B, thus subsuming all feature structures that are at least equally informative as itself. Since it carries no information, the empty feature structure [ ] *subsumes* not only the feature structures (a) and (b), but also any other feature structures including itself. More strictly speaking, the subsumption relation is a partial ordering over feature structures and is defined as follows:[21]

(46) Definition of Subsumption
Given two typed feature structures, FS_1 and FS_2, FS_1 is said to subsume FS_2, written as $A \sqsubseteq B$ if and only if the following conditions hold:

A. Path values Every path **P** which exists in FS_1 with a value of type **t** also exists in FS_2 with a value which is either **t** or its subtype.

B. Path equivalences Every two paths which are shared in FS_1 are also shared in FS_2.

[21] Carpenter (1992: 43) claims that the subsumption relation is a pre-ordering on the collection of feature structures. It is transitive and reflexive, but not anti-symmetric because it is possible to have two distinct feature structures that mutually subsume each other. But these are alphabetic variants.

C. Type ordering Every type assigned by FS_1 to a path subsumes the type assigned to the same path in FS_2 in the type ordering.

Each of the three conditions A, B, and C can be illustrated as below:

4.9.2 Condition A on Path Values

(47) Example satisfying Condition A

$$A\begin{bmatrix} \textit{verb} \\ \text{AGR} \begin{bmatrix} \textit{agr-cat} \\ \text{NUMBER } \textit{singular} \end{bmatrix} \end{bmatrix} \sqsubseteq B\begin{bmatrix} \textit{verb} \\ \text{AGR} \begin{bmatrix} \textit{agr-cat} \\ \text{PERSON } \textit{3rd} \\ \text{NUMBER } \textit{singular} \end{bmatrix} \\ \text{TENSE present} \end{bmatrix}$$

There is only one path in A: <AGR.NUMBER>. This path exists in B and their values are the same.[22] Hence, Condition A is satisfied. Condition B is inapplicable here, since there is no structure sharing in either of the two feature structures. Condition C is satisfied because every type assigned by A to a path is identical with the type assigned to the same path in B. Hence, A subsumes B.

4.9.3 Condition B on Structure Sharing

(48) Case satisfying Conditon B

$$A\begin{bmatrix} \textit{syn-cat} \\ \text{VALENCE} \begin{bmatrix} \textit{val-cat} \\ \text{SPECIFIER} <\boxed{1}\ \text{NP}> \\ \text{COMPS} <> \end{bmatrix} \\ \text{ARG-STR} <\boxed{1}> \end{bmatrix} \sqsubseteq$$

$$B\begin{bmatrix} \textit{syn-cat} \\ \text{ORTH `walks'} \\ \text{HEAD} \begin{bmatrix} \textit{verb} \\ \text{FORM finite} \end{bmatrix} \\ \text{VALENCE} \begin{bmatrix} \textit{val-cat} \\ \text{SPECIFIER} <\boxed{1}\ \text{NP}> \\ \text{COMPS} <> \end{bmatrix} \\ \text{ARG-STR} <\boxed{1}> \end{bmatrix}$$

This example looks a bit complicated. But one can easily check that the structure sharing tagged by $\boxed{1}$ in A also exists in B and the other two conditions are also satisfied. Hence, this subsumption relation holds here.

Consider the following pair of examples related to the structure sharing condition:

[22] The indices A and B are tagged to the features structures for our present discussions only.

(49) Another case involving structure sharing

$$C\begin{bmatrix} word \\ \text{SYN} \begin{bmatrix} syn\text{-}cat \\ \text{HEAD} \begin{bmatrix} noun \\ \text{AGR } \boxed{1}\begin{bmatrix} agr\text{-}cat \\ \text{PERSON } 3rd \end{bmatrix} \end{bmatrix} \\ \text{VALENCE} \begin{bmatrix} val\text{-}cat \\ \text{SPECIFIER} < \begin{bmatrix} noun \\ \text{AGR } \boxed{1} \end{bmatrix} > \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$D\begin{bmatrix} word \\ \text{SYN} \begin{bmatrix} syn\text{-}cat \\ \text{HEAD} \begin{bmatrix} noun \\ \text{AGR} \begin{bmatrix} agr\text{-}cat \\ \text{PERSON } 3rd \end{bmatrix} \end{bmatrix} \\ \text{VALENCE} \begin{bmatrix} val\text{-}cat \\ \text{SPECIFIER} < \begin{bmatrix} noun \\ \text{AGR} \begin{bmatrix} agr\text{-}cat \\ \text{PERSON } 3rd \end{bmatrix} \end{bmatrix} > \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Here, D subsumes C because it satisfies Condition A and C, while Condition B is not relevant. On the other hand, C does not subsume D because Condition B applies here and is violated.

4.9.4 Condition on the Type Ording

This condition applies only to typed feature structures under the assumption of some kind of type inheritance hierarchy assumed. Pronouns, proper nouns, and common nouns are subtypes of the supertype noun. Hence, all these subtypes share some properties of each being a noun. Thus, the following is a simple example of subsumption:

(50) Case involving type ordering

$$A\begin{bmatrix} noun \\ \text{ORTH } \text{‘Mary’} \\ \text{AGR} \begin{bmatrix} \text{PERSON } 3rd \\ \text{NUMBER } singular \end{bmatrix} \end{bmatrix} \sqsubseteq B\begin{bmatrix} name \\ \text{ORTH: } \text{‘Mary’} \\ \text{AGR:} \begin{bmatrix} \text{PERSON } 3rd \\ \text{NUMBER } singular \\ \text{GENDER } feminine \end{bmatrix} \end{bmatrix}$$

where $noun \sqsubseteq name$

Since the type *noun* of A is a supertype of the type *name* of B, A subsumes B. Furthermore, B has an extra piece of information about the gender. Hence, A properly subsumes B.

4.10 Operations on Feature Structures and Feature Values

As a corollary to subsumption, unification is the main topic of this section. Compatible feature structures can be unified to carry a more increased piece of information in general. Generalization is its dual and will also be discussed. In addition, some operations like the concatenation $\oplus$ of list values, alternative feature values and conjunctive types will also be topics of this section.

Kiyong: this section hasn't been completed yet.

4.10.1 Compatibility

Some feature structures are *compatible* with some others, while there are conflicting cases. Consider the following three ßavm's:[23]

(51) a. A
$$\begin{bmatrix} \textit{noun} \\ \text{AGR} \left[\text{PERSON } \textit{3rd}\right] \end{bmatrix}$$

b. B
$$\begin{bmatrix} \textit{noun} \\ \text{AGR} \begin{bmatrix} \text{NUM Sg} \\ \text{GENDER feminine} \end{bmatrix} \end{bmatrix}$$

c. C
$$\begin{bmatrix} \textit{noun} \\ \text{AGR} \begin{bmatrix} \text{PERSON } \textit{3rd} \\ \text{GENDER } \textit{masculine} \end{bmatrix} \end{bmatrix}$$

The feature structure A is compatible with B and also with C. But the feature structures B and C are incompatible because their information about the gender of a noun is conflicting. Incompatibility may also arise when there is a type difference, as shown below:

(52) Incompatible feature structures

a. E
$$\begin{bmatrix} \textit{noun} \\ \text{AGR} \begin{bmatrix} \text{PERSON } \textit{3rd} \\ \text{NUMBER singular} \end{bmatrix} \end{bmatrix}$$

b. F
$$\begin{bmatrix} \textit{verb} \\ \text{AGR:} \begin{bmatrix} \text{PERSON } \textit{3rd} \\ \text{NUMBER } \textit{singular} \end{bmatrix} \end{bmatrix}$$

The feature structures E and F may have the same agreement features, but they are incompatible because their types are different: one is a noun, but the other a verb.

[23] Type labels like *syn-cat*, *val-cat* and *agr-cat* are not very informative, so they will be omitted from now on.

4.10.2 Unification

Compatible feature structures often represent different aspects of information from different sources. Merged together, they may convey a more coherent picture of information. This process of information merge is captured by the operational process of unifying two compatible feature structures, FS_1 and FS_2, represented $FS_1 \sqcap FS_2$.[24] Compatible feature structures can be unified together to form a more (or at least equally) informative feature structure. The unification of two typed feature structures FS_1 and FS_2 is the most general typed feature structure which is subsumed by both FS_1 and FS_2, if it exists.[25]

(53) Formal definition of unification
The unification of $F_1 \sqcap F_2$ of two typed feature structures F_1 and F_2 is the greatest lower bound of F_1 and F_2 in the collection of typed feature structures ordered by subsumption.

The feature structure A, for instance, can be *unified* with C, yielding a little bit more enriched feature structure D.

(54) Unified feature structure

$$D\begin{bmatrix} \textit{noun} & \\ \text{AGR} & \begin{bmatrix} \text{NUMER} & \textit{singular} \\ \text{PERSON} & \textit{3rd} \\ \text{GENDER} & \textit{masculine} \end{bmatrix} \end{bmatrix}$$

Unification normally adds information as illustrated just now. But the identical features may unify without adding any further information. The empty feature structure may unify every feature structure without changing the content of the latter, thus formally treated as the *identity element* of unification.

4.10.3 Unification of Shared Structures

[24]

[25] This and the following definition are copied from Copestake (2002: 55, 61).