ISO TC 37/SC 4 N033 Rev. 1
2003-07-25

Language Resource Management

Descriptors and Mechanisms for Language Resources

File ID SC4N033 doc (479 ko)

SC4N033.pdf (286 ko)

Title:	Draft - Language Resource Management - Feature Structures - Part 1: Feature Structure Representation
Editor(s):	Kiyong Lee
Source:	WG 1
Project number:	24610-1 - This reference will supersede all previous one and remain attached as working ref along the duration of the project.
Status:	CD to be attached to CD ballot
Date:	2003-07-25
Agenda / Action:	For transmission to ISO SC
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

Language Resource Management – Feature Structures

General Comments
Hasida :
Semantics (the graph) of feature structure should be made explicit.

XML may be used more straightforwardly to syntactically represent feature structures, with tags as types and attributes as features. In order to thereby deal with typed feature structures, XML schema can be extended so as to allow multiple inheritance.

We need some guideline as to what should be attributed to schema validation.

Relation with RDF should be discussed.

In this connection, what we may describe by feature structures should be made explicit, too. Probably we will not use them for general knowledge representation, though there is no technical reason preventing us to.

KATS: [We partially agree with Hasida's comments: . . . what we may describe by feature structures should be made explicit, too. Probably we will not use them for general knowledge representation, though there is no technical reason preventing us to.] The title of the document shows that our feature structures are used for the purpose of language resource management. Nevertheless, explicit remarks may be necessary in a concluding section concerning their potential applications.

Lee Gillam:
I'm not certain whether my previous comments made it to the list, but I've added them in below just in case. They can be summarised by Terry Langendoen's comment about "the essentials of feature structure analysis and representation are revealed". The resulting document can be quite 'sterile', but it would be nice to be able to hand it to a software engineer for implementation, so the more clear the rules and restrictions, the better.

As Gary Simons notes, the ISO standards process does include a balloting process, which occurs at various levels for a number of possible documents. There has, however, been no requirement for a reference implementation such as that needed for, for example, Java Specification Request. If the operations, notation, semantics are clear, it should be possible to provide some extent of an implementation, but not necessarily one that is complete (where does one draw the line between the representation in XML and the interpreting software? XSLT is a good case in point).

Language Resource Management – Feature Structures
Part 1: Feature Structure Representation

Table of Contents

Annex B (informative): Basic Operations on Feature Structures

KATS: move the content of Annex B to Section 4.6 and 4.7.

Annex C (normative): Feature Structure DTD

KATS: change Annex C to B

Foreword

(to be filled in)

Introduction

This standard proposal results from the agreement between the Text Encoding Initiative Consortium and ISO committee TC37/SC4 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 one is dedicated to the description of what feature structures are, providing a formal semantics for these, as well as a reference XML-based format for exchanging feature structures between applications. The second one describes one possible way of documenting such structures and expressing constraints on the feature names and feature values that can contain.

KATS: The "formal semantics" mentioned here can be provided in the proposed section 4.9.

KATS: The aim of Part 2 has been understood as providing an implementation standard for XML-based feature structures. But the statement above is not too clear.

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 elementary features, feature values and combination of features, thus offering means to check the conformance of a feature structure with regards to a reference specification.

2 Normative references

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?

KATS: We agree with both of Erjavec's comments.

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.

attribute
property or qualification of some object being described
Note: it is sometimes called *feature* in feature structures. It takes a unique value to form an attribute-value pair called *feature specification* that becomes an element of a feature structure.

Simons: "feature" is defined to mean two different things in two successive sentences! The second I would call a "feature specification", that is, the association of a feature with a particular value is a feature specification. It is then feature specifications (not features) that make up feature structures.

Erjavec :
attribute: I find the discussion here and elsewhere on the status of "feature" confusing. I think the term feature should be mentioned only once, and suggested that it be avoided due to its ambiguity.

KATS: We agree with Simons and Erjavec. The ambiguous term "feature" should be discarded from the main document. Thus, attribute-value pairs should be referred to as feature specifications.

attribute-value matrix
AVM
a very common notation in a matrix form by which a feature structure consisting of attribute-value pairs is represented
Note: In this notation, each row represents a sequence of an attribute and its unique value. Its acronym is AVM.
boxed integer
integer in a box like 1 marking structure sharing in a feature structure

Erjavec : boxed integer: I think it should say that it marks s.s. in an AVM, not in a feature-structure.
KATS: We also agree.

compatibility
two feature structures are compatible if and only if none of the attributes that they have in common has a conflicting value
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 that points to itself
Note: A feature structure is often represented by a dag.

Burnard : Talking of DAGs, I'm not sure that this mechanism can or should support cyclic graphs. There is a casual reference to these in footnote 3 which I think needs expansion, or removal.

Erjavec :
DAGs are of course by definition not cyclical. And FSs are usually DAGs; but there have been some suggestions that you could use cyclical FSs to model some linguistic phenomena; I think Kaspar and Rounds, Karttunen, and Krieger wrote about cyclic FSs. App. B also has a bit in cyclic FSs.

empty path
path corresponding to the root node of a graph
empty feature structure
feature structure that has no attribute-value pairs
Note: It is represented as [ ] and is often called a *variable*.

feature
attribute or property of an object being described
Note: by taking a unique value for the described object, it constitutes part of a feature structure. For this reason, it may often refer to an attribute-value pair, instead of the attribute alone.

Simons: This "Note" is really defining a different thing, namely, feature specification. I think that the precision of terminology required in a standard makes it essential that the term "feature" not be used ambiguously.
E.g.
"Number" is a feature
"Number = sg" is a feature specification
And "sg" is a feature value. That is possibly another term that should be added to the glossary.

KATS: delete Note.

KATS: add
feature specification
attribute-value pair in a feature structure

feature structure
a set of attribute-value pairs carrying partial information about some object being described by assigning a value to each of its attributes.
It is thus defined in set-theoretic terms as a partial function from attributes to values. Because of its mathematical elegance, it is represented in a rooted and directed (acyclic or cyclic) graph. But it creates some typesetting problems when it gets complex. Thus, the matrix notation called AVM often replaces the graph notation. See attribute-value matrix.

Simons: These definitions go back and forth between different set of terminologies. More precisely, I would think that a feature structure is a set of feature specifications, while an attribute-value matrix is a set of attribute-value pairs, and there is a formal equivalence between the two. But it seems imprecise to mix them in the definitions.

KATS: redefine
feature structure
a set of feature specifications.
Note: an attribute-value matrix which represents a feature structure is a set of attribute-value pairs, and there is a formal equivalence between the two.

feature structure declaration
sometimes called *feature structure description.* A feature structure may be described in a declarative manner through some description language.
Simons: If "feature structure declaration" is meant to mean the same thing that it does in TEI P4, then this definition is not right. An FSD does not describe a feature structure. It describes the set of all valid feature structures.

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.

graph notation
A rooted and directed labeled graph is often used to represent a feature structure. Each graph representing a feature structure starts with a single particular node called *the root*. From the root, at least one or more arcs labeled with features branch out to other nodes that again represent appropriate types with their feature structures or terminate as their atomic values.

path
sequence of features that label each arc on a descending sequence of arcs from the root
Note: on a graph, for instance, the root may either remain as the empty path without any branching or may point to other nodes through one or more directed arcs each labeled with a feature.

reentrancy
structure sharing
phenomenon. Through reentrancy, two paths point to the same node on a graph that represents a feature structure. These paths are then called *equivalent*. As a result, the two paths leading to that intersecting node share its features or attribute values. In the AVM notation, reentrancy is conventionally marked by a boxed integer like 3 by tagging it next to the right of the feature structure or the type name of that node and also at the place of the value being shared by the other path without copying the shared feature structure.

root
topmost node on a graph or an (upside-down) tree that has no ancestors nor any (preceding) path.

Erjavec:
root: I would delete 'nor any preceding path' as this follows from it not having any ancestors

shared structure
a feature structure with some attributes sharing values. In graph notation, a node to which two paths merge represents a shared structure. In matrix notation, the shared structure is represented by an identical boxed integer. See *reentrancy*.

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 information in B.

Erjavec: subsumption: this is the only entry where a math symbol is given – as they are not used in TEI part I think it would be best just to remove it (or, otherwise, also introduce it for unification)

KATS: The math symbol for subsumption must be retained, since it is suggested that Annex B dealing with subsumption be incorporated into Section 4.6.

tag
boxed integer marking structure sharing.

Simons: Are you sure you want to introduce this term in a standard about an XML encoding scheme?

Burndard : The draft could point out that the things (very confusingly) called tags in the matrix representation scheme are equivalent to the ID/IDREF mechanism in XML. It might also explain why

the root of an FS in the DAG representation is represented as a type attribute, and so on.

Erjavec :XML. It might also explain why the root of an FS in the DAG

Not really equivalent: coreference means structure sharing so there is no directionality involved like there is with with ID/IDREF; but, yes, you could model it with symmetric ID/IDREF.

KATS: The term tag is used informally in linguistic literature referring to structure sharing. Nevertheless, its technical use may be redefined.

type
some common feature that classify objects in a structured way. Elements of any domain can be sorted into types, based on similarities of properties. In linguistics, for instance, features like *phrase*, *word*, *pos*(parts of speech), *noun*, and *verb* are often taken as types.

Simons: I'm not exactly sure what is meant here, but I think it isn't quite right.

Either "phrase" is the value of a type feature, e.g. "type= phrase" in which case phrase is a feature value, not a feature.

Or, phrase is a feature and its value is a typed feature structure, e.g. "pharse = [FS of type phrase]"

In either case, I don't think it is right to say "the feature phrase is taken as a type".

Erjavec:type: "some common feature that *classify"

Erjavec: “typed feature structures”
As for my comments, the biggest misgiving is what do with types/sorts. The TEI has them, but only as attributes; but types also form hierarchies, and, in some formalisms (like Carpenter) have associated constraints (introduce attributes).

As Carpenter etc did not exist when the TEI was written, all this is right now simply ignored. I think it should be decided whether to a) introduce this extra machinery into the DTD or
b) say in the standard explicitly that type hierarchies and constraints on types are not supported by the standard.

Follow comments on the current draft:

Introduction:

says that the standard is composed of two main parts - the second on
FSDs is still to come?

KATS: Type may be redefined as below,
type
elements of any collection can be sorted into types, based on similarities of their properties.
In linguistics, for instance, categories like *phrase*, *word*, *pos*(parts of speech), *noun*, and *verb* are often taken as types.

KATS: insert
typed feature structure
feature structure that is labeled by a type.

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.

unification
a binary operation on feature structures that combine two compatible feature structures into one representing neither more nor less information than is contained in the feature structures being unified.

Simons:? or "representing exactly all the information"

value
value of a feature in a feature structure may either be atomic or complex. A value is complex if it is a feature structure itself or a list of values, again either atomic or complex.

Simons: Or should this really be made into two entries, one for "atomic value" and one for "complex value"?

4 General characteristics of feature structure

Burnard: The general introductory material on feature structures is useful and informative, especially for the reader who doesn't know about this particular way of representing information. However, it seems very strange that the introduction makes no reference at all to the way in which feature structures may be represented using XML, since that is the topic of the standard! The comparisons between the matrix and tree notations are very helpful, but why not extend them to include comparison with the XML notation?

KATS: We agree with Burnard. Adding an addtional section on the XML notation is required. See below.

Burnard:The introduction also needs a para introducing the idea of feature (etc.) libraries

KATS: We agree with Burnard. The main text of Section 4.8 Type Inheritance may include a brief introduction to the ideas of feature libraries.

Clergerie:
About FS Lite
The revised draft should show either explicitly or through its structure that there are several layers of increasing complexity when using the proposed Feature Structure representation scheme.
For instance, we can identify the following layers:
1. Using untyped feature structures with basic values and no recursion (only need of *fs* without attribute and *f* with attribute *name*). This layer should actually cover most usages where people only want to express a set of basic properties on objects.
2. Adding disjunctions on values (the other kinds of disjunctions may come later)
3. Adding libraries and compact notations
4. Adding types
5. Adding recursion and reentrency
6. Adding complex grouping (alternatives, bags, lists and sets)

7. Adding declaration mechanisms
I think the 3 or 4 first layers should cover most common usages.

KATS: There are two approaches to constructing a theory: One is to go from particulars to generals, and the other is from a general theory to particular cases. We think the second approach is sounder than the first one because the general structure of the theory can be retained in building particular cases.

Overview

KATS: delete the section heading "Overview".

A *feature structure* is a general-purpose data structure that identifies and groups together individual *features*, each of which associates a name with one or more values. 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 feature values to be of specific *types*, and for restrictions to be placed on the values for particular features, by means of *feature system declarations*, which are 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. (See illustration example in non-normative annex 1).

KATS:
Change *features* in line 2 to *feature specifications*
change "annex 1" in the last line to Annex A

Elementary Feature Structures: Features with Binary Values introduces the *binary* feature values, and shows how elementary feature structures using features with those values may be constructed;
Feature, Feature-Structure and Feature-Value Libraries introduces the tags that represent *libraries* of features, feature structures and feature values, along with methods for pointing at features, feature structures and feature values in these libraries;
Symbolic, Numeric, Measurement, Rate and String Values, presents the tags for *symbolic*, *numeric*, *measurement*, *rate*, and *string* values;
Structured Values, shows how to use feature-structures themselves as values, thus enabling feature structures to be recursively defined;
Singleton, Set, Bag and List Collections of Values demonstrates the use of multiple values for features, for encoding *set*, *bag*, and *list* collections of values;
Alternative Features and Feature Values present various methods for representing alternations (disjunctions) of features and feature values;
Boolean, Default and Uncertain Values, presents tags for *boolean*, *default*, and *uncertain* values, along with methods for *underspecifying* feature values;
Indirect Specification of Values Using the rel Attribute shows how to specify various logical relations, such as negation and subsumption, between the expressed values for a feature and its actual values.

KATS: remove paragraph 2 of the overview to the beginning of section 5 as part of its overview.

4.1 Use of Feature Structures

Feature structures may be understood as providing *partial information* about some object described by specifying *values* of some of its *attributes*[1]. Suppose we're describing an employee named Mary

[1] An attribute is often called a 'feature'. Hence, the term 'feature' may refer to an attribute-value pair or and an attribute only. Its dual use must be disambiguated by its context.

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 attributes of hers. These pieces of information can be put into a simple set notation, as in:

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

The use of feature structures can easily be extended to linguistic descriptions, too. Various linguistic features of the word 'love', for instance, can be described by a feature structure of the form:

(2) About the word 'love'
{<PHON, 'love'>, <SYN, {<POS, verb>, <VAL, transitive>}>, <SEM, {<REL, loving}>}.

Here, the attribute PHONology takes a word 'love' as atomic value, whereas the attributes SYNtax and SEMantics take sets of attribute-value pairs as complex value. The complex feature <SYN, <POS, verb>, <VAL, transitive>>, for instance, consists of an attribute SYNtax and its value <POS, verb>, <VAL, transitive>> which is itself a feature structure consisting of two attribute-value pairs[2]. The first type whose value is atomic is called *atomic feature* and the second type whose value is a feature structure *complex feature*.

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.

4.2 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 either as (1) sets of *features* that consist of pairs of *attributes* and their *values* or (2) *labelled directed graphs with a single root* where each arc is labelled with an attribute and directed to its value.

In set-theoretic terms, a feature structure FS can thus be defined as a *partial function* from attributes to values or more formally as a sextuple <*A*, *N*, *r*, *T*,θ, δ> such that

i. *A* is a finite set of attributes.
ii. *N* is a (possibly null) set of nodes.
iii. *r* is a unique member of *N* called *the root*.
iv. *T* is a finite set of types.
v. θ is a function that maps nodes *N* to types *T*.
vi. δ is a partial function from $A \times N$ into *N*.

This definition is general enough to accommodate *typed feature structures* each of which is characterized as being of a particular type or sort. This typing plays a role as constraint on the construction of appropriate feature structures, when a type inheritance hierarchy is specified. By defining feature structures with the value assignment function δ and also with the typing function θ, the unique-value restriction is placed on features: each attribute must be assigned only a single value.

Simons: Ultimately, the solution will also include alternations. For purposes of this definition, is an alternation considered a complex single value?

KATS: yes.

[2] POS stands for part of speech and VAL for valence

Erjavec: “typed feature structures”
As for my comments, the biggest misgiving is what do with types/sorts. The TEI has them, but only as attributes; but types also form hierarchies, and, in some formalisms (like Carpenter) have associated constraints (introduce attributes).

As Carpenter etc did not exist when the TEI was written, all this is right now simply ignored. I think it should be decided whether to a) introduce this extra machinery into the DTD or
b) say in the standard explicitly that type hierarchies and constraints on types are not supported by the standard.

Follow comments on the current draft:

Introduction:

says that the standard is composed of two main parts - the second on
FSDs is still to come?

KATS: It is proposed that a new section be added on type hierarchy as section 4.8

4.3 Notations

There are again several ways of representing feature structures. There are two most common ways of representing them: a graph and a matrix notation.

4.3.1 Graph Notation

For conceptual coherence and mathematical elegance, feature structures are often represented as labelled directed graphs with a single root[3]. The formal definition given earlier can be understood as specifying rooted, directed and labelled arcs on a graph. The attribute-value function δ labels each arc from one node n to another node n' with an attribute name a by mapping a pair (a, n) to the node n': $\delta(a, n) = n'$ or $n \xrightarrow{a} n'$. The typing function θ then maps each node to a type in T, for instance τ to n and τ' to n': $\theta(n) = \tau$, $\theta(n') = \tau'$. These two can then be combined into the following:

Simons: Isn't this backwards? It should be n to tau and n-prime to tau-prime.

KATS: We agree with Simons.

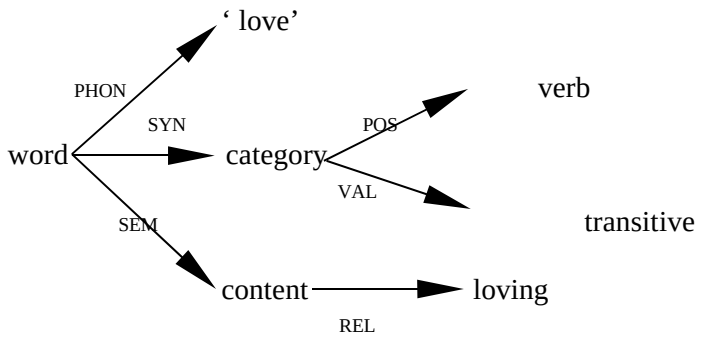
(3) $n: \tau \xrightarrow{a} n' : \tau'$

As stated earlier, values can be either atomic or complex. In a graph notation, atomic values are simple object being represented as *terminal nodes*. Complex values are, on the other hand, feature

[3] 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. Burnard : Talking of DAGs, I'm not sure that this mechanism can or should support cyclic graphs. There is a casual reference to these in footnote 3 which I think needs expansion, or removal.
Erjavec : DAGs are of course by definition not cyclical. And FSs are usually DAGs; but there have been some suggestions that you could use cyclical FSs to model some linguistic phenomena; I think Kaspar and Rounds, Karttunen, and Krieger wrote about cyclic FSs. App. B also has a bit in cyclic FSs.

structures themselves, thus being represented by *non-terminal nodes* that branch out further to other arcs. The partial information about the word 'love' given in (2) can be represented in a graph notation as follows:

(4) Feature structure in graph notation



On this graph, each node is labelled with a type. The root node, for instance, is of a word type, branching out to three nodes whose types are **'love'** , **category**, and **content**, respectively. Each arc is also labelled with an appropriate attribute that is called feature. The arc labelled as PHON is directed to the terminal node **'love'** , an atomic value. The other two arcs labelled as SYN and SEM further branch out: one branches out to the two terminal nodes, **verb** and **transitive**, through the two arcs labelled as POS and VAL. The non-terminal node labelled as SEM is directed to the node **loving** through the arc labelled as REL.

The notion of *path* is useful for reading branches on a graph. A path is a sequence of arc labels or attributes. At the root level, the path is an empty sequence. The path from the root to the terminal node **verb** in the above graph is a sequence <SYN, POS>. Since every graph representing a well-formed feature structure is rooted, the root is reachable from any node.

4.3.2 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. Each feature, or attribute-value pair, in a feature structure is represented as a row with an attribute followed by its value. Note that a colon or a little empty space separates an attribute from its value on each row of an AVM.

(5) Feature structure in an AVM notation

$$\begin{bmatrix} word \\ \text{SURF} \ \text{"love"} \\ \text{SYN} \begin{bmatrix} category \\ \text{POS verb} \\ \text{VAL transitive} \end{bmatrix} \\ \text{SEM} \begin{bmatrix} act \\ \text{REL loving} \end{bmatrix} \end{bmatrix}$$

This example illustrates a feature structure in matrix notation that consists of an atomic feature and

two complex features that take feature structures as their value. The AVM as a whole is of type *word*, while its constituent AVM's are of type *cat* and *content.*

KATS: add 4.3.3 XML Notation

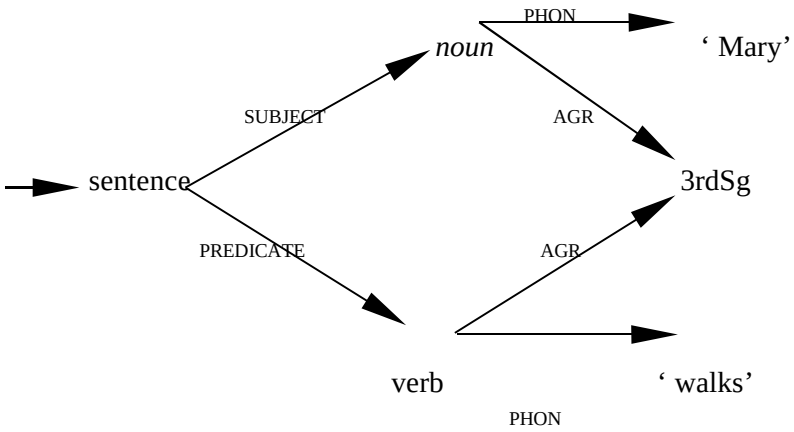
4.4 Shared Feature Structure or Reentrancy

Simons: Delete “Feature”

KATS: We agree.

The graphic notation can clearly represent shared feature values. Consider the following:

(6) Merging paths in graph notation

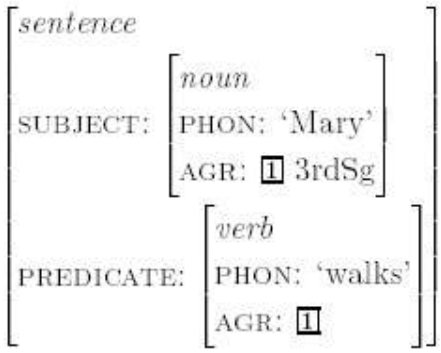


Here, the two AGR paths merge on the node *3rdSg*, indicating these two attributes share one and the same value.

Such sharing can also be represented in an AVM.

$$\begin{bmatrix} \textit{sentence} \\ \text{SUBJECT:} \begin{bmatrix} \textit{noun} \\ \text{PHON: ‘Mary’} \\ \text{AGR: } \boxed{1}\ \text{3rdSg} \end{bmatrix} \\ \text{PREDICATE:} \begin{bmatrix} \textit{verb} \\ \text{PHON: ‘walks’} \\ \text{AGR: } \boxed{1} \end{bmatrix} \end{bmatrix}$$

(7) Shared value in an AVM notation



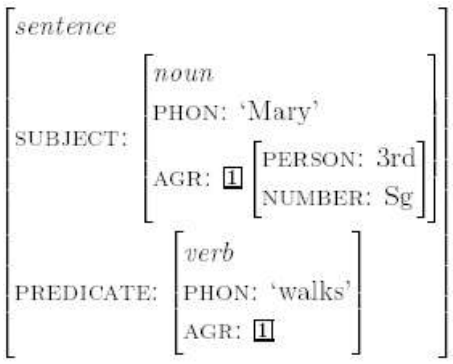
The tags can be attached to feature structures, too.

Simons: "index" would be an alternate term that doesn't overlap with XML terminology. In the next sentence, you could say "tagged with an index to represent value sharing.

KATS: Agreed.

The atomic value *3rdSg* in the above can be expanded to a feature structure and then this feature structure can be tagged to represent value sharing.

(8) Tagging of a feature structure



Tagging has the same effect as the co-indexing in linguistic analysis. For instance, the coindexed nouns in 'Her$_i$ mother loves Mary$_i$' indicate that they are coreferential. This fact can also be easily represented in an AVM by tagging. Note that the token identity among expressions does not guarantee the identity of their values, as in 'Mary's mother loves Mary'.

Erjavec:
"Her *other loves Mary" "mother" I guess.

KATS: Corrected

"Note that token identity .. does not guarantee the identity of their values" <- just the opposite, surely!

KATS: We agree. Change the "token identity" in the text above to "type identity".

4.5 List Values

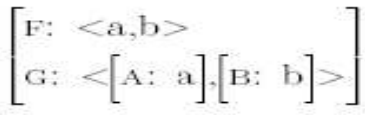
KATS: change Section title to: 4.5 Set, Bag, and List as Feature Values

Burnard : In section 4,5 there is discussion only of simple lists. Since the TEI scheme goes to some length to distinguish lists, bags, and sets (inter alia), it might be worth mentioning that not all lists are simple here!

KATS: We agree.

The domain of complex values can be extended to accommodate lists of atomic values or feature structures as attribute values. The following could be an example:

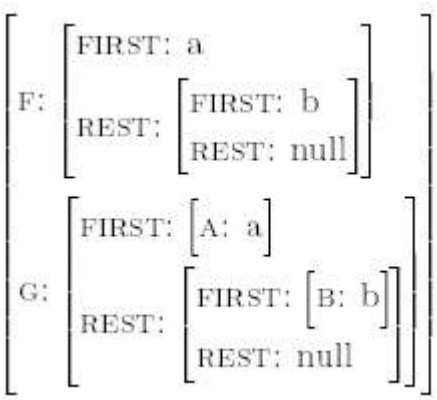
(9) List as an attribute value



Note that a list as an attribute value may consist of either atomic or complex values.

List values can also be represented recursively as shown below:

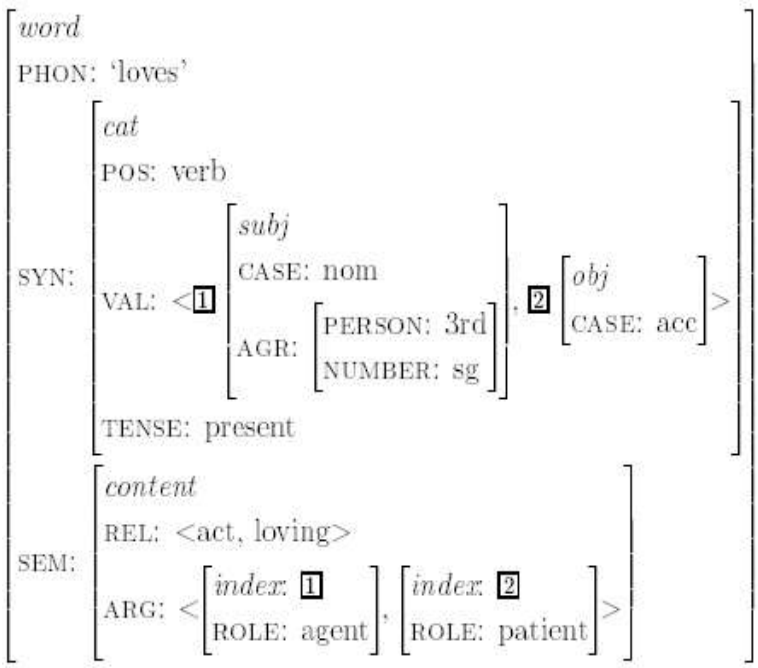
(10)Recursive representation



List values are useful for treating complement, valence or argument structures of predicates in syntax or semantics. The internal structure of a verb 'loves' can be represented in more detail as follows[4].

(11) Valence taking a list as its value

[4] Linguists may not agree on the linguistic descriptions presented here. These are only here for the sake of illustration.



This feature structure contains three lists of values: one is a list of atoms, as in the feature REL: <act, loving>, while the other two are lists of feature structures, each representing the valence structure or the argument structure of the verb 'loves'. The two elements in these lists are each linked to the other by tagging.

Erjavec : I would suggest moving Appendix B into end of 4 - it has the same status and it is strange to have an intro to FSs without mentioning unification, even though the standard itself does not use it.

KATS: We agree with Erjavec.
KATS: add 4.6 Relations on Feature Structures (Subsumption)
KATS: add 4.7 Operations on Feature Structures (Unification and Generalization)
KATS: add 4.8 Type Inheritance
KATS: add 4.9 Semantics of Feature Structures

5 Feature Structure representation

KATS: insert Overview, part of which will be the second paragraph removed from the Overview of section 4.

Burnard: There are a couple of comments saying that more linguistic examples are needed. There are quite a few of these in the TEI vault, which I would be willing to dig out and bring up to date with the current P4 syntax if that would be of use.

Clergerie:
About atomic types:
I don't believe that a draft about Feature Structures should enumerate a list of atomic types. Such a list would overlap with specification of atomic types in other place and would never be complete. For instance, someone may wish at some point to have dates as atomic type or any kind of formatted strings. XML schema are an alternative where

atomic types may be defined. The FS draft should rather focus on how to use atomic types.

Clergerie:
About reentrency
The current draft does not precise how reentrency is handled. One may think that XML references (through IDs) are a solution. However, I am afraid they are not, because of *renaming* problems and because of the use of FS libraries. Indeed, the reentrency points present in several occurrences of a same feature structure (from a library) and used in different places should be considered as distinct.
Xpointer notation seems to be a better alternative.

```
<fsLib>
   <fs id ="fs1" >
      <f name="f"><  fs id="fs2"></fs></f>
      <f name="g">
   <fs id ="fs3">
      <f name="h" sharing =" . . / . . / f [ @name=f ] " / >
      . . .
   </fs>
   #</fs>
   . . .
</fs Lib>
```

Notes:
- should look for a linguistic example
- not sure about the best notation (an attribute *sharing* of *f*) or some new element inside *f*

An alternate and more readable notation exists, not using Xpointer, is possible.

```
<fs Lib>
   <fs id ="fs1" >
      <f name="f" var="X"><fs id ="fs2"> . . . < /fs ></f>
      <f name="g">
   <fs id ="fs3" >
      <f name="h" var ="X"/>
     . . .
</fs>
</fs>
. . .
</fs Lib>
```

Footnote:
This notation has the advantage of beeing symmetric but says nothing about the scope of variable X. For simple cases, it may be assumed that it is the topmost feature structure containing the variable but this is not a fully acceptable answer.

KATS: not sure of the implication or consequences the change might cause if the suggestion is adopted.

5.1 Elementary Feature Structures: Features with Binary Values [16.2]

Erjavec :it seems strange to list all possible values of <f> but not to mention that a <fs> can be a value as well.

KATS: mention <fs> as a possible value of <f> as well

The fundamental elements of a feature structure system are **<f>** (for *feature*) and **<fs>** (for *feature structure*). The **<fs>** element has a **type** attribute for indicating what type of feature structure it represents, and may contain any number of **<f>** elements. An **<f>** element, in turn, has a required **name** attribute and any number of associated *values*. These may be binary, numeric, symbolic (i.e. taken from a restricted set of legal values), or string-valued, or may consist of sets, lists, or bags of binary, numeric, symbolic, or string values. Specialized values may also be given which allow partial underspecification of the feature. These possible types are all described in more detail in this and the following sections.
This section considers the special case of feature structures that contain features whose single value is one of the *binary values* represented by the empty elements **<plus>** and **<minus>**. The elements that are used for representing feature structures, features and the binary values, along with their descriptions and attributes, are the following.

- **<fs>** analyzes a collection of features and feature alternations as a structural unit.

  type provides a type for a feature structure.
  feats pointer to features.
  rel indicates the relation of the given content to the actual content or value of the feature structure.
- **<f>** associates a name with a value of any of several different types.

  name provides a name for a feature.
  org indicates organization of given value or values as singleton, set, bag or list.
  fVal points to the **id** attributes of feature values.
  rel indicates the relation between the values that are given as the content of the feature or pointed at by the **fVal** attribute and the actual values of the feature.
- **<plus>** provides binary plus value for a feature.

  No attributes other than those globally available (see definition for a.global)

Simons: I really doubt that we want to carry the set of global attributes from TEI into this standard.

I find it hard to imagine a circumstance under which <plus> could have any attributes at all. Even id doesn't make sense. There is only one global concept of "plus", and each instance of <plus> is a reference to the one global concept.

LeeGillam:

On the subject of 'plus' or 'null', and other attributes(?) on such elements (?), perhaps a look to RDF gives a pointer? If 'plus' is defined as a universal concept, would a specific 'plus' be an instance of this? This would seem to make sense from an RDF perspective.

Is 'id' then an attribute that has a 'domain' of 'element'? (With a caveat that it is not used on instances of universal concepts)? On a similar note, would it make sense to define the 'domain' of 'lang' as an element (feature, or feature-structure? I feel 'element' and 'attribute' are both unclear at this point), and leave it at that? From a keep-it-simple perspective, is there any additional benefit in preventing this association, or could an implementation not just ignore it? Something

to spark debate on a Monday!

In reply to this email and another, rather than putting comments through BSI currently, I'd prefer to give a view of the document as a whole, which you have made a good start at bringing into ISO.

The closest standard I am aware of in relation to it is ISO 11404, Information technology - Programming languages, their environments and system software interfaces - Language-independent datatypes. The relationship is more to do with the way in which certain items have been described. e.g (although it does not reproduce so well here):

- <**minus**> provides binary minus value for a feature.

  No attributes other than those globally available (see definition for a.global)

The attributes not discussed in this section are discussed in following sections as follows: the **feats** and the **fVal** attributes in section 5.2 [16.3] Feature, Feature-Structure and Feature-Value Libraries, the **rel** attribute in section 5.8 [16.9] Indirect Specification of Values Using the rel Attribute, and the **org** attribute in section 5.5 [16.6] Singleton, Set, Bag and List Collections of Values.

An **<fs>** element containing **<f>** elements with binary values can be straightforwardly used to encode the *matrices* of feature-value specifications for phonetic segments, such as the following for the English segment [s].

```
+---           ---+
| + consonantal   |
| - vocalic       |
| - voiced        |
| + anterior      |
| + coronal       |
| + continuant    |
| + strident      |
+---           ---+
```

Using the additional tag set for feature structures, this might be encoded as follows. Note that **<fs>** elements may have a **type** attribute indicating the kind of feature structure in question.

```
<fs type="phonological segment">
   <f name="consonantal"> <plus/>  </f>
   <f name="vocalic">     <minus/> </f>
   <f name="voiced">      <minus/> </f>
   <f name="anterior">    <plus/>  </f>
   <f name="coronal">     <plus/>  </f>
   <f name="continuant">  <plus/>  </f>
   <f name="strident">    <plus/>  </f>
</fs>
```

The restriction of specific features to specific types of values (e.g. the restriction of the feature ‘strident' to the values **<plus/>** or **<minus/>**) cannot be validated by a generic XML parser (though other validation mechanisms such as XML Schemas do provide such capabilities). To enable an application program to check that only legal values for particular features appear, one may write a *feature-system declaration*, as described in section 6 (Feature System Declaration representation).

5.2 Feature, Feature-Structure and Feature-Value Libraries [16.3]

As the example in the preceding section illustrates, the direct encoding of features structures can be verbose. Consequently, the effort of encoding large numbers of feature structures in this manner could be enormous, and could result in the creation of enormous files. To reduce the size and complexity of the task of encoding feature structures, one may use the **feats** attribute of the **<fs>**

element to point to one or more of the features of that element. This indirect method of encoding feature structures presumes that the **<f>** elements are assigned unique **id** values, and are collected together in **<fLib>** elements (*feature libraries*). In turn, feature structures can be collected together in **<fsLib>** elements (*feature-structure libraries*). Finally, one may use the **fVal** attribute of the **<f>** element to point to its values. This indirect method of encoding feature values presumes that the value elements are assigned **id** specifications, and are collected together in **<fvLib>** elements (*feature-value libraries*). The elements which are used for representing feature, feature-structure and feature-value libraries, along with their descriptions and attributes, are the following.

- <**fLib**> assembles library of feature elements.

  type indicates type of feature library (i.e., what kind of features it contains).

Simons: <fs> has a formally defined type attribute, and the formal definition of feature structures in section 4 provides a formal definition of type for feature structures, thus the "type" attribute on fsLib has a meaningful definition that is consistent with everything that has preceded.

But what does "type" mean in reference to an <f> or to a feature value that is not an <fs>? If it means anything, it does not mean the same thing is what has been formally defined as type, thus the "type" attribute should not be used.

KATS: see below

- <**fsLib**> assembles library of feature structure elements.

  type indicates type of feature-structure library (i.e., what type of feature structures it contains).
- <**fvLib**> assembles library of feature value elements.

  type indicates type of feature-value library (i.e., what type of feature values it contains).

For example, suppose a feature library for phonological feature specifications is set up as follows.

```
<fLib type="phonological features">
   <f id="CNS1" name="consonantal"> <plus/> </f>
   <f id="CNS0" name="consonantal"> <minus/> </f>
   <f id="VOC1" name="vocalic">     <plus/> </f>
   <f id="VOC0" name="vocalic">     <minus/> </f>
   <f id="VOI1" name="voiced">      <plus/> </f>
   <f id="VOI0" name="voiced">      <minus/> </f>
   <f id="ANT1" name="anterior">    <plus/> </f>
   <f id="ANT0" name="anterior">    <minus/> </f>
   <f id="COR1" name="coronal">     <plus/> </f>
   <f id="COR0" name="coronal">     <minus/> </f>
   <f id="CNT1" name="continuant">  <plus/> </f>
   <f id="CNT0" name="continuant">  <minus/> </f>
   <f id="STR1" name="strident">    <plus/> </f>
   <f id="STR0" name="strident">    <minus/> </f>
   <!-- ... -->
</fLib>
```

Then the feature structures that represent the analysis of the phonological segments (phonemes) `/t/`, `/d/`, `/s/`, and `/z/` can be defined as follows.

```
<fs feats="CNS1 VOC0 VOI0 ANT1 COR1 CNT0 STR0"/>
<fs feats="CNS1 VOC0 VOI1 ANT1 COR1 CNT0 STR0"/>
<fs feats="CNS1 VOC0 VOI0 ANT1 COR1 CNT1 STR1"/>
<fs feats="CNS1 VOC0 VOI1 ANT1 COR1 CNT1 STR1"/>
```

The preceding are but four of the 128 logically possible fully specified phonological segments using the seven binary features listed in the feature library. Presumably not all combinations of features correspond to phonological segments (there are no strident vowels, for example). The legal combinations, however, can be collected together in a *feature-structure library*, with each element being given a unique **id** attribute, as in the following example.

```
<fsLib id="fsl1" type="phonological segment definitions">
```
Simons: I see by this example that type on <fsLib> means something different from type on <fs>. Therefore, do NOT use the same term and attribute. What is meant here is an adhoc grouping, not a formally defined type.

Possible alternatives: kind=, group=, set=,

KATS: "group" sounds fine

```
   <!-- ... -->
   <fs id="T.DF" feats="CNS1 VOC0 VOI0 ANT1 COR1 CNT0 STR0"/>
   <fs id="D.DF" feats="CNS1 VOC0 VOI1 ANT1 COR1 CNT0 STR0"/>
   <fs id="S.DF" feats="CNS1 VOC0 VOI0 ANT1 COR1 CNT1 STR1"/>
   <fs id="Z.DF" feats="CNS1 VOC0 VOI1 ANT1 COR1 CNT1 STR1"/>
   <!-- ... -->
</fsLib>
```

Text elements can be linked to these feature structures in any of the ways described in section 15.2 Global Attributes for Simple Analyses of the TEI guidelines. In the following example, a **<linkGrp>** element is used to link selected characters in the text 'Caesar seized control' to their phonological representations.

```
<text id='TXT1'>
  <!-- ... -->
  <body>
      <!-- ... -->
      <ab id='S1'>
        <w id='S1W1'><c id='S1W1C1'>C</c>ae<c id='S1W1C2'>s</c>ar</w>
            <w   id='S1W2'><c   id='S1W2C1'>s</c>ei<c   id='S1W2C2'>z</c>e<c
id='S1W2C3'>d</c></w>
        <w id='S1W3'>con<c id='S1W3C1'>t</c>rol</w>.
      </ab>
      <!-- ... -->
  </body>
  <fsLib id='FSL1' type='phonological segment definitions'>
      <!-- as in previous example -->
  </fsLib>
  <linkGrp type='phonological identification of characters'
           domains='FSL1 TXT1'
           targFunc='phonological.segment character' >
      <!-- ... -->
      <link id='LT' targets='T.DF S1W3C1'/>
      <link id='LD' targets='D.DF S1W2C3'/>
      <link id='LS' targets='S.DF S1W2C1'/>
      <link id='LZ' targets='Z.DF S1W2C2'/>
      <!-- ... -->
  </linkGrp></text>
```

Simons: Shouldn't these be T.DF and D.DF?

Because of the simplicity of the binary feature values, there is no particular gain in pointing at those values rather than specifying them directly. However, the mechanism of using the **fVal** attribute on **<f>** elements is useful for representing more complex feature values, and can be illustrated using binary values. Suppose the **<plus>** and **<minus>** elements are collected together in a **<fvLib>**, as follows.

Simons: If one did use fVal to refer to a complex value, then it would be out of an fsLib, not an fvLib!

I personally don't see the justification for the fvLib. When two <f>s have different ids, they are different feature specifications. When two atomic values have different ids, they are not different values!!!! <plus id="mine"> means exactly the same thing as <plus id="yours">.

However, <f id="mine" ...> and <f id="yours" ...> can mean something different. Thus, it is legitimate to point to <fs> and <f> instances, but not to primitives.

This example with fVal="B1" doesn't save any space over simply embedding <plus>, nor does it add any functionality, but it does add to the complexity of what anybody implementing software needs to implement to support the standard.

Even with something like <str> values which might be long, so that one might argue that pointing to values in a fvLib would be justified, it is still dubious because there is essentially a one-to-one mapping from feature specification to value. That is, just wrap the values in the FVLib with the <f> they go with, and you now have the fLib. As far as I can tell, the fvLib just adds cost to implementers without a comensurate amount of benefit.

KATS: sounds correct. We just get away with fvLib, incorporating its content into fLib and revising the relevant text and the examples accordingly.

```
<fvLib type="binary values">
   <plus id="B1"/>
   <minus id="B0"/>
</fvLib>
```

Then the feature library presented at the beginning of this section can be represented as follows.

```
<fLib type="phonological features">
   <f id="CNS1" name="consonantal" fVal="B1"/>
   <f id="CNS0" name="consonantal" fVal="B0"/>
   <f id="VOC1" name="vocalic" fVal="B1"/>
   <f id="VOC0" name="vocalic" fVal="B0"/>
   <f id="VOI1" name="voiced" fVal="B1"/>
   <f id="VOI0" name="voiced" fVal="B0"/>
   <f id="ANT1" name="anterior" fVal="B1"/>
   <f id="ANT0" name="anterior" fVal="B0"/>
   <f id="COR1" name="coronal" fVal="B1"/>
   <f id="COR0" name="coronal" fVal="B0"/>
   <f id="CNT1" name="continuant" fVal="B1"/>
   <f id="CNT0" name="continuant" fVal="B0"/>
   <f id="STR1" name="strident" fVal="B1"/>
   <f id="STR0" name="strident" fVal="B0"/>
   <!-- ... -->
</fLib>
```

Although **<fs>** elements are legitimate feature values (see section 5.4 [16.5] Structured Values), they are not allowed within **<fvLib>** elements. They should be placed in **<fsLib>** elements.

5.3 Symbolic, Numeric, Measurement, Rate and String Values [16.4]

Erjavec : "This library would have a total of 1620 (3 9 3 5 2 2) entries."
missing × !

KATS:

End: as Lou notes, not only <rate> but <str> has been dropped as well - I'd suggest putting both in again.

KATS: should be discussed by the contributors

In section 5.1 [16.2] Elementary Feature Structures: Features with Binary Values, we defined the two empty elements **<plus>** and **<minus>** which are used to represent binary values. In this section, we

define five more feature-value elements: the empty elements **<sym>** for expressing *symbolic values*, **<nbr>** for expressing *numeric values*, **<msr>** for expressing *measurement values*, and **<rate>** for expressing *rate values*; and the element **<str>** for expressing *string values*. These elements, along with their descriptions and attributes, are the following.

- **<sym>** provides symbolic values for features.

  value provides a symbolic value for a feature, one of a finite list that may be specified in a feature declaration.
  Rel indicates the relation of the given value to the actual value.
- **<nbr>** provides a numeric value or range of values for a feature.

  value provides a numeric value.
  valueTo together with **value** attribute, provides a range of numeric values.
  type indicates whether value or range is to be understood as real or integer.
  Simons: insert "valueTo" instead of "range"
  Rel indicates the relation of the given value or range to the actual value or range.
- **<msr>** provides a measure value or range of values for a feature.

  unit provides a unit for a measure feature, one of a finite list that may be specified in a feature declaration.
  value provides a numeric value.
  valueTo together with **value** attribute, provides a range of numeric values.
  type indicates whether value or range is to be understood as real or integer.
  Simons: insert "valueTo" instead of "range"
  Rel indicates the relation of the given value or range to the actual value or range.
- **<rate>** provides a rate value or range of values for a feature.

Burnard : A large section explaining not only <rate> but also <str> has been excised from section 5.3. I understand why <rate> might not seem immediately relevant to linguistic applications (tho surely there might be some applications in phonology?) but I think it should be kept in the standard, and that means it needs to be explained as clearly as the other primitives.

Erjavec :"rel" attribute is everywhere mistyped as "Rel"
This section is missing a big chunk on <valRange>:
The <sym> element is to be used for the value of a feature when that feature can have any of a small, finite set of possible values, representable as character strings. </gap>

  unit provides a unit for a rate feature, one of a finite list that may be specified in a feature declaration.
  per provides an interval for a rate feature, one of a finite list that may be specified in a feature declaration.
  value provides a numeric value.
  valueTo together with **value** attribute, provides a numeric range of values.
  type indicates whether value is to be understood as real or integer.
  Simons: insert "or valueTo" after "value"
  Rel indicates the relation of the given value or range to the actual value or range.
- **<str>** provides a string value for a feature.

Rel indicates the relation of the given value to the actual value. The **<sym>** element is to be used for the value of a feature when that feature can have any of a small, finite set of possible values, representable as character strings.
[Titre 3]Example
Features with **<sym>**, **<plus>**, and **<minus>** values may be used to encode highly structured information such as may be obtained from precoded survey instruments. We illustrate by means of a coding scheme based on the one that is used for classifying potential printed entries in the British

National Corpus. The scheme uses the following features and associated values.

medium

books and magazines; miscellaneous; written to be spoken

domain

imaginative; applied science; arts; belief and thought; commerce and finance; leisure; natural and pure science; social science; world affairs

level

high; medium; low

sampling range

beginning; middle; end; whole; whole less ten percent

date of origination

1960–1975; 1975–1993

published (miscellaneous items only)

yes; no

selection method (books and periodicals only)

chosen on grounds of circulation or influence; chosen at random

A comprehensive feature library for this scheme is the following; the **id** specifications are those used by the British National Corpus (BNC) project:

```
<fLib type="BNC classification features">
 <f id="ca002" name="medium"><sym value="book.or.periodical"/></f>
 <f id="ca003" name="medium"><sym value="miscellaneous"/></f>
 <f id="ca004" name="medium"><sym value="written.to.be.spoken"/></f>
 <f id="ca005" name="domain"><sym value="imaginative"/></f>
 <f id="ca006" name="domain"><sym value="applied.science"/></f>
 <f id="ca007" name="domain"><sym value="arts"/></f>
 <f id="ca008" name="domain"><sym value="belief.and.thought"/></f>
 <f id="ca009" name="domain"><sym value="commerce.and.finance"/></f>
 <f id="ca00a" name="domain"><sym value="leisure"/></f>
 <f id="ca00b" name="domain"><sym value="natural.and.pure.science"/></f>
 <f id="ca00c" name="domain"><sym value="social.science"/></f>
 <f id="ca00d" name="domain"><sym value="world.affairs"/></f>
 <f id="ca00e" name="level"><sym value="high"/></f>
 <f id="ca00f" name="level"><sym value="medium"/></f>
 <f id="ca00g" name="level"><sym value="low"/></f>
 <f id="ca00h" name="sample.type"><sym value="beginning"/></f>
 <f id="ca00j" name="sample.type"><sym value="middle"/></f>
 <f id="ca00k" name="sample.type"><sym value="end"/></f>
 <f id="ca00l" name="sample.type"><sym value="whole"/></f>
 <f id="ca00m" name="sample.type"><sym value="whole.less.ten.percent"/></f>
 <f id="ca00n" name="published.between"><sym value="1960.1975"/></f>
 <f id="ca00p" name="published.between"><sym value="1975.1993"/></f>
 <f id="ca00r" name="published"><plus/></f>
 <f id="ca00s" name="published"><minus/></f>
 <f id="ca00t" name="selection.method"><sym value="principled"/></f>
 <f id="ca00u" name="selection.method"><sym value="random"/></f>
</fLib>
```

An entry which is a book or periodical on world affairs, medium level, sampled from the middle, published between 1975 and 1993, and selected on a principled basis could then be assigned the following feature-structure code; this code could also be placed in a feature-structure library that contains all the possible fully-specified BNC entry classifications. This library would have a total of 1620 (3 9 3 5 2 2) entries.

```
<fs   id="ca2dfjpt"
      type="BNC classification for written documents"
      feats="ca002 ca00d ca00f ca00j ca00p ca00t"/>
```

[Note: an example for <alt> should be provided here]

[Note: the following examples should be more closely related to language resource

management]

The **<nbr>** element is to be used when the value of a feature is a number or a range of numbers. For example, suppose one wishes to encode information contained in classified advertisements for the sale or rental of real estate, such as the number of bedrooms and bathrooms in a listed property, and its advertised selling or rental price. One way of representing such information is as follows.

```
<fs type="real estate listing">
   <f name="number.of.bathrooms"><nbr value="2"/></f>
   <f name="number.of.bedrooms"><nbr value="3"/></f>
   <f name="monthly.rent"><nbr value="625.00"/></f>
</fs>
```

The information that the number of bedrooms is in the range from 3 to 5 and the monthly rent is in the range from 625.00 to 950.00 may be represented as follows, using the optional **valueTo** attribute.

```
<fs type="real estate listing">
   <f name="number.of.bedrooms"><nbr value="3" valueTo="5"/></f>
   <f name="monthly.rent"><nbr value="625.00" valueTo="950.00"/></f>
</fs>
```

The **<nbr>** (and also the **<msr>** and **<rate>** elements defined below) element also may have a **type** attribute to specify whether the values of the **value** and **valueTo** attributes are to be construed as integer or real numbers.

The **<msr>** element to be used when the value of a feature is a scalar quantity, essentially a combination of a numeric value and a symbolic value for identifying the scale on which the numeric value occurs. For example, real estate listings often provide the area (in square feet or meters) of a house or apartment and the area (in acres or hectares) of land being sold or rented. One way of representing information about such areas is as follows.

```
<fs type="real estate listing">
   <f name="interior.area"><msr value="2000" unit="sq.ft"/></f>
   <f name="property.area"><msr value="0.5" unit="acre"/></f>
</fs>
```

The value of the ‘monthly.rent' feature in the two examples above might be more accurately analysed as a measurement rather than as a numeric value, since the amount of the rent in question is to be understood as payable in a specific currency (US or Canadian dollars, pounds sterling, euro, yen...) To make the currency scale explicit, the first example of this feature might be re-encoded as follows.

```
<f name="monthly.rent"><msr value="625.00" unit="USD"/></f>
```

The **unit** and **value** attributes of the **<msr>** element are both required. If the **unit** attribute is not needed (for example, if no confusion would result if the **unit** attribute is not specified), then the **<nbr>** element may be used to express the feature value.

[Note: the original description of <rate> has been dropped]

Simons: Does that mean <rate> is being dropped from the proposed standard? That wouldn't bother me, but if so, it should be dropped from the opening paragraph of this section and the list of elements and attributes.

5.4 Structured Values [16.5]

Features may have *structured values* as well; these values are represented by either the **<fs>** element, or the **fVal** attribute on the **<f>** element, which can point to an **<fs>** element. Since an **<fs>** or a pointer to an **<fs>** is permitted to occur as a value of an **<f>**, recursion is possible. For example, an **<fs>** element may contain or point to an **<f>** element, which may contain or point to an **<fs>** element, which may contain or point to an **<f>** element, and so on. To illustrate the use of structured values, consider the following simple model of a personal record, consisting of a person's name, date of birth, place of birth, and sex. Each personal record is a `<fs type='personal record'>` tag, consisting of the corresponding four features, three of which take structured values, as in the following example.

[LR -> KL: an example more closely related to language resources should be provided here]

```
<fs type="personal record">
```

```
  <f name="full.name">
    <fs type="name record">
      <f name="first.name"> <str>Kathleen</str> </f>
      <f name="middle.name"> <str>Anne</str> </f>
      <f name="surname"> <str>Barnett</str> </f>
    </fs>
  </f>
  <f name="date.of.birth">
    <fs type="date record">
      <f name="year"> <nbr value="1968"/> </f>
      <f name="month"> <nbr value="4"/> </f>
      <f name="day"> <nbr value="17"/> </f>
    </fs>
  </f>
  <f name="place.of.birth">
    <fs type="place record">
      <f name="city"> <str>Austin</str> </f>
      <f name="state"> <sym value="TX"/> </f>
    </fs>
  </f>
  <f name="sex"> <sym value="female"/> </f>
</fs>
```

Now suppose that feature-structure libraries are maintained for name records and place records. Further suppose that the feature structure representing the name record in the previous example has an **id** attribute with the value `nkab027`, while the feature structure representing the place record has an **id** attribute whose value is `txaustin`.[5] Then the preceding example could also be encoded as follows. (An identifier is also provided for the personal record.)

```
<fs id="pkab027" type="personal record">
   <f name="full.name" fVal="nkab027"/>
   <f name="date.of.birth">
      <fs type="date record">
         <f name="year"> <nbr value="1968"/> </f>
         <f name="month"> <nbr value="4"/> </f>
         <f name="day"> <nbr value="17"/> </f>
      </fs>
   </f>
   <f name="place.of.birth" fVal="txaustin"/>
   <f name="sex"> <sym value="female"/> </f>
</fs>
```

This representation could be simplified further if a feature library is maintained for the year, month, day and sex features, so that the **feats** attribute may be used as follows.

```
<fs id="pkab027" type="personal record" feats="sxf">
   <f name="full.name" fVal="nkab027"/>
      <f name="date.of.birth"><fs type="date record" feats="y1968 m04
d17"/></f>
   <f name="place.of.birth" fVal="txaustin"/>
</fs>
```

Next, suppose that a feature-structure library is also maintained for personal records, and that the library also contains records for the parents of the individual identified in the previous example. Suppose that the father is identified as `pmfb009` and the mother as `parn002`. Then the personal-record feature structure could be easily augmented to include pointers to the parents, as follows.

```
<fs id="pkab027" type="personal record" feats="sxf">
   <f name="full.name" fVal="nkab027"/>
      <f name="date.of.birth"><fs type="date record" feats="y1968 m04
d17"/></f>
```

[5] [Rem.: should not this be placed in-line somewhere in the standard]: Feature-structure, rather than feature-value, libraries should be used for housing collections of feature structures.

```
   <f name="place.of.birth" fVal="austintx"/>
   <f name="mother" fVal="parn002"/>
   <f name="father" fVal="pmfb009"/>
</fs>
```

If the personal records identified as `parn002` and `pmfb009` also contain information about the parents of those individuals, then from the present record, one would have access to that individual's grandparents as well.

Assuming that personal records of the sort described in this section are being maintained in association with text files, the records can be linked to those texts in any of the ways described in chapter 14 Linking, Segmentation, and Alignment of the TEI guidelines, provided that identifiers are added for appropriate features, as in the following illustration.

```
<text id="bfile"><body>
    <div id="tkab027" type="birth certificate">
      <p><name id="t1kab027" type="person">Kathleen Anne Barnett</name>
        was born at <time id="t1t0659">6:59 a.m.</time> on
        <date id="t1d680417">April 17, 1968</date> in
        <name id="t1setonhsp" type="org">Seton Hospital</name> in
        <name id="t1txaustin" type="place">Austin</name> to
        <seg id="s1">Mr.</seg> and <seg id="s2">Mrs.</seg>
        <name id="t1mfb009" type="person">Michael F. Barnett</name>
        of <name id="t1sansabatx" type="place">San Saba</name>.
      </p>
      <!-- ... -->
      <join id="t1arn002" targets="s2 t1mfb009"/>
      <join id="t2mfb009" targets="s1 t1mfb009"/>
      <!-- ... -->
    </div></body>
  <fsLib id="prec" type="personal records">
    <fs id="pkab027" type="personal record" feats="sxf">
      <f name="full.name" fVal="nkab027"/>
      <f id="dkab027" name="date.of.birth">
        <fs type="date record" feats="y1968 m04 d17"/>
      </f>
      <f id="bkab027" name="place.of.birth" fVal="txaustin"/>
      <f id="mkab027" name="mother" fVal="parn002"/>
      <f id="fkab027" name="father" fVal="pmfb009"/>
    </fs></fsLib>
  <linkGrp type="record verification" domains="bfile prec" targFunc="source
goal">
    <link targets="t1kab027 nkab027"/>
    <link targets="t1d680417 dkab027"/>
    <link targets="t1txaustin bkab027"/>
    <link targets="t1arn002 mkab027"/>
    <link targets="t2mfb009 fkab027"/>
  </linkGrp>
</text>
```

5.5 Singleton, Set, Bag and List Collections of Values [16.6]

Simons: There is a slight incongruity here in that the introduction to feature structures in section 4 only talks about lists. And in the literature on feature structures in computational lingusitics, that is about all you run into. It is probably worth considering simplifying the new standard to support just list, and even to do it by a <list> element contained within an <f>, rather than by an "org" attribute.

KATS: it is suggested that bag & set be introduced in chapter 4. Dropping bags and sets altogether would require discussion among contributors

Erjavec :
Mistakes in P4?
"In a set, items are ordered, and may not be repeated."
surely "not ordered"!?
"The <null> element when used with a feature organised as a singleton is a semantic error; however, its appearance as a value for such a feature cannot be flagged by SGML or XML parsers."
I'd imagine it is simple to find such cases using a schema language?

KATS: sure, “not ordered”

In the discussion to this point, we have assumed that features have exactly one simple value. However, for many purposes, it is useful to be able to consider the values of certain features to be organized in more complex ways, for example as sets, bags (or multisets), or lists. Accordingly, we provide for four different ways in which feature values may be organized, namely as *singletons*, *sets*, *bags* and *lists*. We do so by means of an **`org`** attribute on the **`<f>`** element, which takes on one of the designated values `single`, `set`, `bag`, and `list`. A feature whose value is organized as a singleton is understood as having exactly one simple value. If more than one value is specified for it, we assume that only the first one is considered to be its true value. A feature whose value is organized as a set, bag or list may have any positive number of values as its content. In a set, items are ordered, and may not be repeated. In a bag, items are not ordered, and may repeat. In a list, items are ordered and may repeat. Sets and bags are thus distinguished from lists in that the order in which the values are specified does not matter for the former, but does matter for the latter, while sets are distinguished from bags and lists in that repetitions of values do not count for the former but do count for the latter.[6] No default value for the **`org`** attribute is declared in the DTD; however, a default value for that attribute can be declared for particular features in the feature-system declaration; see section 6 [26] Feature System Declaration. Note that if only one value is specified for a given **`<f>`** element, the set, bag and list values of the **`org`** are all essentially equivalent to the singleton value, so the omission of the **`org`** attribute for such a feature is not problematic.[7]
To illustrate the use of the **`org`** attribute, suppose that the illustration of personal records from the previous section is extended to include pointers to an individual's siblings. Suppose also that the individual identified as `<fs id="pkab027">` has siblings identified as `<fs id="panb005">`, `<fs id="pmfb010">` and `<fs id="pzrb001">` in the personal records library. Then we may extend the personal record for `<fs id="pkab027">` as follows.

```
<fs id="pkab027" type="personal record" feats="sxf">
   <f name="full.name" fVal="nkab027"/>
   <f name="date.of.birth">
      <fs type="date record" feats="y1988 m04 d17"/>
```

[6] An XML DTD cannot however straightforwardly validate that values for features organized as sets are not repeated;

Simons: It is possible in an XML Schema with a unique constraint. Would there also be a Schema for the standard?

KATS:

such validation would have to be carried out by an application program. Our method of representing set, bag and list values also does not permit such values to be directly embedded within one another. In order to embed a set within a set, for example, one must specify the embedded set as the value of a feature of a feature-structure value of the including set. Fortunately, this is not as hard as it sounds: the embedding of a list within a list is illustrated in the second example below.

[7] Unless the value is the <null> element; see below.

```
   </f>
   <f name="place.of.birth" fVal="austintx"/>
   <f name="mother" fVal="parn002"/>
   <f name="father" fVal="pmfb009"/>
   <f name="siblings" org="set" fVal="panb005 pmfb010 pzrb001"/>
</fs>
```

A more elaborate illustration of the use of the **org** attribute is the following `<f name="career" org="list">` element which may be added to the personal records of an individual to record the job career of that individual. The feature structures that constitute the value of this feature document the jobs which the individual has held in the order in which they were held. Note that a list has been embedded within a list by means of intervening `<fs type="employment record">` and `<f name="promotion.history">` elements.

```
<f name="career" org="list">
  <fs type="employment record">
    <f name="employer"><str>Safeway Stores</str></f>
    <f name="hiring.information">
      <fs type="hire structure">
              <f name="hire.date"><fs type="date structure" feats="y1988
m06"/></f>
        <f name="job.title"><sym value="stocker"/></f>
        <f name="wage"><rate value="6.00" per="hour"/></f>
        <f name="hours.worked"><rate value="40" per="week"/></f>
        <f name="status.code" fVal="sc4a"/>
      </fs>
    </f>
    <f name="promotion.history" org="list">
      <fs type="promotion record">
        <f name="date"><fs type="date structure" feats="y1988 m12"/></f>
        <f name="job.title"><sym value="cashier"/></f>
        <f name="wage"><rate value="7.00" per="hour"/></f>
        <f name="hours.worked"><rate value="40" per="week"/></f>
        <f name="status.code" fVal="sc4a"/>
      </fs>
      <fs type="promotion record">
        <f name="date"><fs type="date structure" feats="y1990 m02"/></f>
        <f name="job.title"><sym value="supervisor"/></f>
        <f name="salary"><rate value="18000" per="year"/></f>
        <f name="status.code" fVal="sc3c"/>
      </fs>
    </f>
    <f name="termination.information">
      <fs type="termination structure">
         <f name="termination.date"><fs type="date structure" feats="y1991
m04"/></f>
        <f name="status.code" fVal="sc3c"/>
        <f name="reason.for.termination"><sym value="laid.off"/></f>
      </fs>
    </f>
  </fs>
  <fs type="employment record">
    <!-- ... -->
  </fs>
  <!-- ... -->
</f>
```

The information contained in such features may be linked to textual references in the usual way. The `<f name="status.code">` feature has been included to show how evaluative or interpretive information can be included along with information gleaned from textual records. The example presumes that the status code values are maintained in a designated **`<fvLib>`**.

Features with values organized as sets, bags or lists can sometimes be used instead of features organized as singletons, whose values are individual feature structures. For example, consider the following encoding of the English verb form ‘sinks', which contains an ‘agreement' feature whose value is a feature structure which contains ‘person' and ‘number' features with symbolic values.

```
<fs type="word structure">
  <!-- ... -->
  <f name="word.class"> <sym value="verb"/> </f>
  <f name="tense"> <sym value="present"/> </f>
  <f name="agreement">
    <fs type="agreement structure">
      <f name="person"> <sym value="third"/> </f>
      <f name="number"> <sym value="singular"/> </f>
    </fs>
  </f>
  <!-- ... -->
</fs>
```

If one does not care about the names of the features contained within the ‘agreement' feature structure, the containing `<f name="agreement">` element can be given an **`org`** attribute with the value `set`, and the contained **`<fs>`** element, together with the person and number feature elements it contained, can be eliminated, as follows.

```
<fs type="word structure">
  <!-- ... -->
  <f name="word.class"> <sym value="verb"/> </f>
  <f name="tense"> <sym value="present"/> </f>
          <f     name="agreement"     org="set"><sym     value="third"/><sym
value="singular"/></f>
  <!-- ... -->
</fs>
```

The encoding in the preceding example presumes that the **`<fDecl>`** element for the ‘agreement' feature would look something like the following; for further details, see section 6 [26] Feature System Declaration.

```
<fDecl name="agreement" org='set'>
  <!-- ... -->
  <vRange>
    <vAlt>
      <sym value='first'/>
      <sym value='second'/>
      <sym value='third'/>
    </vAlt>
    <vAlt>
      <sym value='singular'/>
      <sym value='plural'/>
    </vAlt>
  </vRange>
  <!-- ... -->
</fDecl>
```

The set, bag or list which has no members is known as the null (or empty) set, bag or list. To refer to it, the **`<null>`** element is provided; its description and attributes are as follows.

- <**null**> represents the null set, bag, or list, depending on whether the org attribute of its parent f has the value set, bag, or list; has no interpretation if the org attribute of its parent f element has the value single.

  No attributes other than those globally available (see definition for a.global)

Simons: Note that if lists were done with a <list> element, then <null> would not be needed.

It would be represented by <list/>. Thus we could simplify (and bring things more into line with standard practice as described in section 4) by removing the org attribute and <null>, and introducing <list> in their place.

So, for example, to indicate that the individual identified above by the `<fs id="pkab027">` element has no siblings, we may specify the 'siblings' feature as follows.

```
<f name="siblings" org="set"> <null/> </f>
```

The **<null>** element when used with a feature organized as a singleton is a semantic error; however, its appearance as a value for such a feature cannot be flagged by XML parsers. The **<null>** element, when it appears as a feature value, must be the only value.

Simons: Like plus and minus, <null> shouldn't be allowed to have any attributes

KATS: agreed

5.6 Alternative Features and Feature Values [16.7]

Note: It was intended that the representation based on the generic <alt> mechanism be dropped. However, considering that we will need such a generic mechanism for TC37 as a whole, some more in depth thought should be had before just getting rid of this in this section. Someone should make a synthesis on alternation and go through section 14.8 of the guidelines…

In this section, two methods of representing the alternation (ambiguity or uncertainty) of features and feature values are presented. The first of these methods is to be used for nonsystematic or sporadic markup of alternation of individual features or values; it makes use of the special-purpose **<fAlt>** and **<vAlt>** elements. The other is to be used for systematic markup of alternation and for the alternation of groups of features or values; it makes use of the general-purpose **<alt>** element introduced in section 14.8 Alternation of the TEI guidelines (see www.tei-c.org). The **<fAlt>** and **<vAlt>** elements have the following description and attributes.

- **<fAlt>** provides alternative features for a feature structure or other feature alternation.

  mutExcl indicates whether values are mutually exclusive.

- **<vAlt>** provides alternative (disjunctive) values for a feature.

  mutExcl indicates whether values are mutually exclusive.

Burnard: I don't quite understand what is meant by the reference to <alt>. It is a generic mechanism in P4, which could be documented in the ISO standard with reference to alternation in general, as a generalisation of the specific <vAlt> <fAlt> etc. elements. The reason for having these more specific tags, by the way, is that they permit more constrained content models.

Simons: I think this is another place to simplify for the sake of producing an implementable standard. Though we all understand the distinction from formal logic, I don't think the distinction is made in the practice of using feature structures. I don't think I've ever seen feature structures in a linguistics publication that had two different notations for mutually exclusive alternation versus non mutually exclusive; one only every sees a single notation for alternation, thus I think we should only define "alternation" as is the convention in practice.

To illustrate the use of the **<fAlt>** element to represent the alternation of features, suppose one is uncertain whether a particular real estate advertisement describes a house with two bedrooms or with two bathrooms. This uncertainty can be represented as follows.

KATS: no objection

```
<fs type="real estate listing">
   <fAlt>
      <f name="number.of.bathrooms" > <nbr value="2"/> </f>
      <f name="number.of.bedrooms"> <nbr value="2"/> </f>
   </fAlt>
</fs>
```

This representation leaves unspecified whether or not the alternation is *mutually exclusive* (i.e. whether having two bathrooms excludes the possibility of having two bedrooms and vice versa). To make this aspect of the alternation explicit, one can specify a value for the **`mutExcl`** attribute, as follows.

```
<fs type="real estate listing">
   <fAlt mutExcl="N">
      <f name="number.of.bathrooms" > <nbr value="2" /> </f>
      <f name="number.of.bedrooms" > <nbr value="2" /> </f>
   </fAlt>
</fs>
```

The **`<fAlt>`** element can also be used to represent uncertainty about whether the number of bathrooms is two or three, as follows; note that the attribute value **`mutExcl="Y"`** can be inferred for the **`<fAlt>`** element in this example.

```
<fs type="real estate listing">
   <fAlt>
      <f name="number.of.bathrooms" > <nbr value="2" /> </f>
      <f name="number.of.bathrooms" > <nbr value="3" /> </f>
   </fAlt>
</fs>
```

Since the ‘number.of.bathrooms' feature in this example can be factored out of the alternation, a **`<vAlt>`** element could be used in place of it to represent the alternation of the feature values more simply, as follows:

```
<fs type="real estate listing">
   <f name="number.of.bathrooms" >
      <vAlt>
         <nbr value="2" />
         <nbr value="3" />
      </vAlt>
   </f>
</fs>
```

The **`<fAlt>`** and **`<vAlt>`** elements can also be used to indicate certain alternations among values of features organized as sets, bags or lists. For example, suppose one uses a `<f name="extras" org="set">` element in feature structures for real estate listings to represent items that are mentioned to enhance a property's sales value, such as whether it has a pool or a good view. Now suppose for a particular listing, the extras include an alarm system and a fenced-in yard, and either a pool or a jacuzzi (but not both). This situation could be represented, using the **`<vAlt>`** element, as follows.

Simons: I'm not really keen on this. Another example was given above in the example that showed how you could model agreement as a set feature as opposed to a feature structure. It is true that the power of the TEI FSD makes it possible to do this, but do any publications about language description actually do it? What is happening is that we are using the facilities in the TEI FSD to define a regular expression language for defining validity over lists of values. But I think that is a place where feature creep got the better of us--that is not a standard approach to feature sructures, so why define and implement it?

In the agreement case above, I don't think we would want to encourage anybody to do it that way. Straight feature structures without a regular expression mechanism has all the power that is needed to describe agreement. So the regular expression thing does not add descriptive

power; it just adds alternative ways of doing the same thing that adds to the cost of implementation and the non-interoperability of conceptually equivalent data sets.

In this pool versus jacuzzi example, another way to handle it would be in the constraints part of the FSD. that is, to say that if <f name+"extras"> contains pool, it may not also contain jacuzzi. Thus again, removing the regular expression power here does not deprive us of the power to express the correct linguistic facts.

```
<fs type="real estate listing">
   <!-- ... -->
   <f name="extras" org="set" >
      <str>alarm system</str>
      <str>fenced-in yard</str>
      <vAlt mutExcl="Y">
         <str>pool</str>
         <str>jacuzzi</str>
      </vAlt>
   </f>
   <!-- ... -->
</fs>
```

Simons: I may have misunderstood this example in the comment to the right. I was assuming this was defining a class of listings, as opposed to a specific listing. This was because of the wording "but not both". Does this actually mean, the specific house we are describing has either a pool or a jacuzzi, but we aren't sure which it is, but we are sure that it is not both? If that is it, then my comment to the right is somewhat off base since I was commenting from the point of view of an FSD, but the comment is still worth passing on as it relates to using this sort of thing in an FSD.

Now suppose the situation is like the preceding except that one is also uncertain whether the property has an alarm system or a fenced-in yard, or possibly both. This can be represented as follows.

```
<fs type="real estate listing">
   <!-- ... -->
   <f name="extras" org="set" >
      <vAlt mutExcl="N">
         <str>alarm system</str>
         <str>fenced-in yard</str>
      </vAlt>
      <vAlt mutExcl="Y">
         <str>pool</str>
         <str>jacuzzi</str>
      </vAlt>
   </f>
   <!-- ... -->
</fs>
```

Finally, suppose that the listing specifies that the property has a finished basement, and that it also has either an alarm system and a pool or a fenced-in yard and a jacuzzi. This situation cannot be represented using the **<vAlt>** element, because the alternation holds between subsets of two values each. It can, however, be represented using the **<fAlt>** element, as follows; note that the **<str>** element with the value `finished basement` element must be repeated.

```
<fs type="real estate listing">
   <!-- ... -->
   <fAlt mutExcl="Y">
      <f name="extras" org="set" >
         <str>finished basement</str>
```

```
         <str>alarm system</str>
         <str>pool</str>
      </f>
      <f name="extras" org="set" >
         <str>finished basement</str>
         <str>fenced-in yard</str>
         <str>jacuzzi</str>
      </f>
   </fAlt>
   <!-- ... -->
</fs>
```

If a large number of ambiguities or uncertainties involving a relatively small number of features and values need to be represented, it is recommended that the general-purpose **<alt>** element discussed in section 14.8 Alternation be used, rather than the special-purpose **<fAlt>** and **<vAlt>** elements. The use of the **<alt>** element avoids the need to explictly represent the alternating elements more than once.

For example, suppose one has set up a **<fsLib>** element containing feature structures representing the morphological structures of classical Greek inflected words, along with collections of individual features and feature values, encoded by **<fLib>** and **<fvLib>** elements as appropriate. The following example shows how one might then represent the morphological structure of a feminine gender, accusative case, plural number noun form in classical Greek, such as '????' goddesses discussed in section 5.3 [16.4] Symbolic, Numeric, Measurement, Rate and String Values:

```
<fsLib type="noun structures">
   <!-- ... -->
   <!-- plural accusative feminine noun -->
   <fs id="wngfkanp" type="noun structure" feats="wn gf ka np"/>
   <!-- ... --> </fsLib>

<fLib type="morphological features">
   <f id="wn" name="word.class"  fVal="nn"/>
   <!-- ... -->
   <f id="gf" name="gender"  fVal="fe"/>
   <!-- ... -->
   <f id="ka" name="case"  fVal="ac"/>
   <!-- ... -->
   <f id="np" name="number"  fVal="pl"/>
   <!-- ... --> </fLib>

<fvLib type="morphological feature values">
   <!-- ... -->
   <sym id="nn" value="noun" />
   <!-- ... -->
   <sym id="fe" value="feminine" />
   <!-- ... -->
   <sym id="ac" value="accusative" />
   <!-- ... -->
   <sym id="pl" value="plural" />
   <!-- ... --> </fvLib>
```

Simons:This is a good example of the basic one-to-one relationship between feature values and features which I cited above in arguing that fvLib was only adding complexity for implementation with commensurate added benefit. This example becomes fewer bytes if the feature values are embedded in the <f>s rather than pointed to, since the feature values are not getting reused due to the one-to-one from value to feature

Now consider the noun form '????' goddesses, which is analyzable as a feminine plural noun form in either the nominative or the vocative case. We may represent this ambiguity by adding the following entries to the **<fsLib>**, **<fLib>**, and **<fvLib>** elements in the preceding example; assume that appropriate entries for unambiguous nominative and vocative case forms have already been entered.

```
<!-- Add the following to the feature-structure library  -->
   <!-- plural nominative-or-vocative feminine noun -->
   <fs id="wngfknvnp" type="noun structure" feats="wn gf knv np"/>
<!-- Add the following to the feature library  -->
   <!-- CASE='nominative' or vocative -->
   <f id="knv" name="case"  fVal="novo"/>
<!-- Add the following to the feature value library -->
   <!-- nominative or vocative -->
   <alt id="novo" targets="no vo"/>
```

If the **<fvLib>** element is not used, and specifications for particular feature values are entered as content of the **<f>** elements in the **<fLib>** element, then the ambiguity can be represented as follows.

```
<fsLib type="noun structures">
   <!-- ... -->
   <!-- plural nominative-or-vocative feminine noun -->
   <fs id="wngfknvnp" type="noun structure" feats="wn gf knv np"/>
   <!-- ... -->
</fsLib>
<fLib type="morphological features">
   <!-- ... -->
   <f id="kn" name="case" >
      <sym value="nominative" />
      <!-- ... -->
   </f>
   <f id="kv" name="case" >
      <sym value="vocative" />
      <!-- ... -->
      <alt id="knv" targets="kn kv"/>
      <!-- ... -->
   </f>
</fLib>
```

The **<alt>** element together with the **<join>** element can, unlike the **<fAlt>** and **<vAlt>** elements, be used to express alternations between sets of features. An example of such an alternation is found in certain feminine gender Greek noun forms ending in -??, such as ?????? attempt(s), which may be analyzed as having either genitive case and singular number features or accusative case and plural number features, as follows (again, assuming the existence of other elements and identifier attributes for simple features and values).

```
<!-- Add the following to the feature structure library -->
   <!-- feminine noun, either genitive singular or accusative plural -->
   <fs id="wngfkg.nska.np" type="noun structure" feats="wn gf kg.nska.np"/>
<!-- Add the following to the feature library  -->
   <join id="kg.ns" targets="kg ns"/><!-- genitive singular -->
   <join id="ka.np" targets="ka np"/><!-- accusative plural -->
   <!-- alternation:  gen. sg. or acc. plural -->
   <alt id="kg.nska.np" targets="kg.ns ka.np"/>
```

5.7 Boolean, Default and Uncertain Values [16.8]

Erjavec : "(see definition for a.global)"

this is not provided in the current version - opens the question how to make hooks into the TEI P4 - as to a (normative!) reference, or have the standard self-contained (difficult!).

In this section we define four special empty elements used as feature values: the *boolean value* elements **<any>** and **<none>**, the **<dft>** element, and the **<uncertain>** element.
The boolean value elements are used to indicate whether the features they are associated with have values. The element **<any>** corresponds to the boolean value `true` (i.e., that the feature it is associated with has a value —not the same as the binary value `plus`), and the element **<none>** corresponds to the boolean value `false` (i.e., that the feature it is associated with has no value). The **<dft>** element is used to indicate that the feature it is associated with has its default value in the feature structure in which it appears. Finally, the **<uncertain>** element may be used to indicate uncertainty about what value, if any, its associated feature has; it is equivalent to the alternation of the **<any>** and **<none>** elements. To indicate uncertainty about which of the possible legal values a particular feature has, one should use the **<any>** element.
The descriptions and attributes of these elements are as follows.

- <**any**> represents boolean true value variable.

  No attributes other than those globally available (see definition for a.global)

Simons: Delete "No attributes other than …"4 places.
<any> <none>

This might be true in some philosophical sense, but it does not seem a useful definition in the context of feature structures. Rather, it is more like:

<any> represents the presence of any possible valid value of the feature

<none> represents the absence of any value for the feature

- <**none**> represents boolean false value variable.

  No attributes other than those globally available (see definition for a.global)
- <**dft**> provides default value for a feature.

  No attributes other than those globally available (see definition for a.global)
- <**uncertain**> provides uncertainty value for a feature.

  No attributes other than those globally available (see definition for a.global)

The values **<null>** and **<none>** are distinct. The former is to be used with a feature organized as a set, bag, or list to indicate that its value is the null set, bag, or list in a particular feature structure. The latter is to be used with such a feature to indicate that it has no value in a particular feature structure.
The *boolean* values **<any>** and **<none>** are also distinct from the *binary* values **<plus>** and **<minus>**. The latter pair are specific possible values for features, whereas the former pair represent *ranges* of possible values, not specific possible values, for features. For example, suppose that the **<valRange>** element for the 'auxiliary' feature is declared as follows in the feature structure declaration, so that either boolean value is legal.

```
<vRange><vAlt><plus/><minus/></vAlt></vRange>
```

Simons: Hmmm. This example shows something that is wrong with the current DTD. A binary feature should always be plus versus minus, or presence or absence, but it seems a weakness that the current system allows people to actually use it either way. Perhaps the FSD should declarea feature as binary, and that automatically means a certain way of using <plus> and <minus>.

Given this **<vRange>**, then the following pair of specifications is distinct:

```
<f name="auxiliary"><plus/></f>
<f name="auxiliary"><any/></f>
```

In this situation, the **<any>** element is equivalent to the alternation of the **<plus>** and **<minus>** values.

Given the same **<vRange>**, then the following pair of specifications is also distinct.

```
<f name="auxiliary"><minus/></f>
<f name="auxiliary"><none/></f>
```

The **<none>** element is equivalent to the negation of the alternation of the **<plus>** and **<minus>** elements.

However, if the auxiliary feature is declared to take only the **<plus>** value, then the specifications below are equivalent:

```
<f name="auxiliary"><plus/></f>
<f name="auxiliary"><any/></f>
```

If the auxiliary feature is declared to take only the **<plus>** value, then the specifications below are not equivalent; in fact, the specification is invalid.

```
<!-- invalid! -->
<f name="auxiliary"><minus/></f>
<f name="auxiliary"><none/></f>
```

It is even possible to declare that a particular feature can never have values, as follows for the ‘impossible' feature:

```
<vRange><null/></vRange>
```

In this case, the following specifications are equivalent.

```
<f name="impossible"><any/></f>
<f name="impossible"><none/></f>
```

The elements **<any>** and **<dft>** are also designed to be used in conjunction with the **<fDecl>** and **<valDefault>** elements in the feature system declaration discussed in section 6 [26] Feature System Declaration. First, consider the **<any>** element, and suppose that the **<vRange>** element in the **<fDecl>** element for the ‘gender' feature is specified as follows.

```
<vRange>
  <vAlt>
    <sym value='feminine'/>
    <sym value='masculine'/>
    <sym value='neuter'/>
    </vAlt>
  </vRange>
```

Then the following two representations are equivalent.

```
<f name="gender"> <any/> </f>
<f name="gender">
   <vAlt>
      <sym value="feminine"/>
      <sym value="masculine"/>
      <sym value="neuter"/>
   </vAlt>
</f>
```

Second, consider the **<dft>** element, and suppose that the default value for the ‘gender' feature (as specified by the **<valDefault>** element of its **<fDecl>** element) is `feminine`. Then the following three representations are equivalent; note that if an **<f>** element appears without content and without a valid **fVal** attribute, then it is equivalent to the same element with the **<dft>** element as its content.

```
<f name="gender"/>
<f name="gender"> <dft/> </f>
<f name="gender"> <sym value="feminine"/> </f>
```

Using the **<any>** and **<dft>** elements, together with an **<fDecl>** element for the corresponding feature in the feature system declaration, provides a method for *underspecifying* the value of that feature. The **<any>** element means that the associated feature has a legal value but what value it has is not specified. The **<dft>** element means that the associated feature has the value which the

encoder has declared is the normal value of the feature.
The boolean elements **<any>** and **<none>** also have specific uses within **<fsConstraints>** and **<fDecl>** elements in feature system declarations, as described in section 6 [26] Feature System Declaration.

Simons: Delete “boolean”

For example, the element **<any>** can appear as the value of a feature contained within an **<fs>** of a particular type which appears in the **<cond>** element of an **<fsConstraints>** element, to indicate that the feature must appear in feature structures of the designated type (i.e., that it is obligatory) and that when it does appear, it may appear with any of its legal values. Similarly, **<none>** can appear in this way to specify that the feature cannot be present in feature structures of the indicated type (i.e., that it is obligatorily absent from such feature structures). All other features that are declared to have values are understood to be optional in such feature structures.
For example, the following may appear as part of the **<fsConstraints>** of a feature system declaration to indicate that an ‘agreement structure' feature structure must contain a legal ‘number' feature, but must not contain a ‘category' feature.

```
<cond> <fs type='agreement structure'></fs>
   <then/><fs>
           <f name='number'><any/></f>
           <f name='category'><none/></f>
         </fs>
</cond>
```

Further constraints can be imposed on a feature structure of a particular type in the **<vRange>** elements of features which take feature structures of that type as values. For example, suppose that verb and adjective agreement in German are represented by feature structures of the following sorts, in which verb forms agree in person and number with their subjects and adjective forms agree in gender, case, and number with their subjects.

```
<fs type="verb structure">
   <!-- ... -->
   <f name="verbAgreement">
      <fs type="agreement structure">
         <f name="person"> <sym value="first"/> </f>
         <f name="number"> <sym value="plural"/> </f>
      </fs>
   </f>
   <!-- ... -->
</fs>
<fs type="adjective structure">
   <!-- ... -->
   <f name="adjagreement">
      <fs type="agreement structure">
         <f name="gender"> <sym value="feminine"/> </f>
         <f name="case"> <sym value="accusative"/> </f>
         <f name="number"> <sym value="plural"/> </f>
      </fs>
   </f>
   <!-- ... -->
</fs>
```

In order to ensure that an ‘agreement structure' feature structure which appears as the value of a ‘verbAgreement' feature may be specified for any person and number feature, but for no gender and case feature, we may provide a **<vRange>** element for the ‘verbAgreement' feature as follows.

```
<vRange>
  <fs type='agreement structure'>
    <f name='person'><any/></f>
    <f name='case'><none/></f>
```

```
    <f name='gender'><none/></f>
    <f name='number'><any/></f>
    </fs>
  </vRange>
```
Similarly, to ensure that an ‘agreement structure' feature structure which appears as the value of a ‘adjAgreement' feature may be specified for any case, gender, and number feature, but for no person feature, we may provide a **<vRange>** element for the ‘adjAgreement' feature as follows.
```
<vRange>
  <fs type='agreement structure'>
    <f name='person'><none/></f>
    <f name='case'><any/></f>
    <f name='gender'><any/></f>
    <f name='number'><any/></f>
    </fs>
  </vRange>
```
The combination of declarations like these and the principle of *subsumption* discussed in section 5.8 [16.9] Indirect Specification of Values Using the rel Attribute, allows feature structures to be underspecified in text markup. For example, to indicate that a given adjective inflection feature (tagged `<f name="adjInflection">`) is a feature structure (tagged `<fs type="inflection structure">`) specifying plural number and any gender and case, we can omit the elements for gender and case on the **<fs>** element, as follows.
```
<f name="adjinflection">
   <fs type="inflection structure">
      <f name="number"> <sym value="plural"/> </f>
   </fs>
</f>
```
When supplied as the value of a ‘verbInflection' feature, the same feature structure would be interpreted as an inflection structure specifying plural number and any person.
If an optional feature is not specified in a feature-structure value, then it is assumed to occur with the **<uncertain>** value. For further discussion, see section 5.8 [16.9] Indirect Specification of Values Using the rel Attribute.

5.8 Indirect Specification of Values Using the rel Attribute [16.9]

Simons: This whole area needs to be carefully re-evaluated. When we were developing the TEI guidelines, we came up with the rel attribute is a way to handle negation (which is typically found in practice). However, by ending up with a relationship attribute, rather than a <not> element, we suddenly opened the door to thinking of all kinds of relationships that could be supported. The result is a system that goes well beyond what any known implementation of feature system software supports.

For instance, the TEI system allows eq, ne, sb, and ns to be specified on feature structues, feature specifications, and values. The literature and implemented systems would just have a negation operator (=ne). I don't thhk examples in the literature would have sb and ns at the primitive value level. At the feature structure level, the eq versus sb relationship is not explicitly signaled--subsumption just gets invoked in certain operations over whole feature structures (but it is not selectively encoded into parts of feature structures).

Note that <f rel="sb"></f> means the same thing as <f><any/></f>. Thus, we don't really need rel="sb" since we already have <any/>

And then when we throw in alternations, sets, bags, and lists we have a real mess for knowing how to interpret all the relations. I'm not sure the full TEI system is well-enough defined formally to implement, for instance, subsumption of a feature whose value is not equal to an alternative involving sets or values that don't subsume something else.

Time to simplify!

The **rel** attribute is provided for the feature value elements **<sym>**, **<nbr>**, **<msr>**, **<rate>**, **<str>**, **<fs>**, and **<default>** (but not **<plus>**, **<minus>**, **<null>**, **<vAlt>**, **<any>**, **<none>**, and **<uncertain>**). This attribute may be used for specifying which of various logical relations the given value has to the actual value of the feature. For all value elements for which the **rel** attribute is defined, except for **<fs>**, the default value for that attribute is eq, which means that the actual value is equal (or identical) to the given value. Accordingly, the following representations are both interpreted to mean that the value of the 'case' feature is the <sym value="genitive"> element.

```
<f name="case"> <sym value="genitive"/> </f>
<f name="case"> <sym rel="eq" value="genitive"/> </f>
```

5.8.1 The Not-Equals Relation [16.9.1]

The **rel** attribute can also be specified as having the value ne, which means that the associated feature has a value which is not equal to the given value. For example, the value <nbr rel="ne" value="1"> in the following example denotes any numeric value other than 1 for the feature 'number.of.bathrooms'.

```
<f name="number.of.bathrooms"> <nbr value="1" rel="ne"/> </f>
```

If an **<fDecl>** element has been provided which defines the legal values for the associated feature, then the value ne can be given a positive interpretation. For example, suppose that the **<vRange>** element is declared in the **<fDecl>** element for the 'case' feature as follows.

```
<vRange>
  <vAlt>
    <sym value='nominative'/>
    <sym value='genitive'/>
    <sym value='dative'/>
    <sym value='accusative'/>
    <sym value='vocative'/>
    </vAlt>
  </vRange>
```

Suppose also that the 'case' feature is declared as obligatory in a particular feature structure. Then the following specifications are equivalent in that structure.

```
<f name="case"> <sym value="genitive" rel="ne"/> </f>
<f name="case">
   <vAlt>
      <sym value="nominative"/>
      <sym value="dative"/>
      <sym value="accusative"/>
      <sym value="vocative"/>
   </vAlt>
</f>
```

That is, when the **rel** attribute occurs with the value ne in the value of an obligatory feature in a feature structure, the actual value of that feature may be any of its legal values *other than* the specified value.

On the other hand, if the 'case' feature is declared as optional in a particular feature structure, then the following specifications are equivalent in that structure.

```
<f name="case"> <sym value="genitive" rel="ne"/> </f>
<f name="case">
   <vAlt>
      <sym value="nominative"/>
      <sym value="dative"/>
      <sym value="accusative"/>
      <sym value="vocative"/>
```

```
      <none/>
   </vAlt>
</f>
```

That is, when the **rel** attribute has the value ne in the value of an optional feature in a feature structure, the actual value of that feature may be any of its legal values other than the specified value, or **<none>**.

If the **rel** attribute is specified with the value ne for a **<nbr>**, **<msr>**, or **<rate>** element for which the **valueTo** attribute is also specified, then the actual range may be any range distinct from that given. For example, the following means that the number of bathrooms is a range distinct from 3 to 5 (e.g., 3 to 4, 3 to 6, 4 to 5, 4 to 6, 0 to 2, etc.).

```
<f name="number.of.bathrooms"> <nbr value="3" valueTo="5" rel="ne"/> </f>
```

Lee Gillam **:Section 6.3.1 Equality**

In every value space there is a notion of equality, for which the following rules hold:

for any two instances (a, b) of values from the value space, either a is equal to b, denoted a = b, or a is not equal to b, denoted a /=b ;

there is no pair of instances (a, b) of values from the value space such that both a = b and a /=b for every value a from the value space, a = a;

for any two instances (a, b) of values from the value space, a = b if and only if b = a;

for any three instances (a, b, c) of values from the value space, if a = b and b = c, then a = c.

On every datatype, the operation Equal is defined in terms of the equality property of the value space, by:

for any values a, b drawn from the value space, Equal(a,b) is true if a = b, and false otherwise.

5.8.2 Other Inequality Relations [16.9.2]

For the elements **<nbr>**, **<msr>**, **<rate>**, and **<str>**, the **rel** attribute may also take on the following values; the use of these values for the **<str>** element presumes that a particular character and string ordering (or *sorting*) convention is understood.

lt
: The actual value or range is any legal value or range less than the specified value or range.

le
: The actual value or range is any legal value or range less than or equal to the specified value or range.

gt
: The actual value or range is any legal value or range greater than the specified value or range.

ge
: The actual value or range is any legal value or range greater than or equal to the specified value or range.

These attribute values may be used as shown in the following examples. The first states that the number of bedrooms is less than 5; the second that an illegal speed is any speed greater than 65 miles per hour; the third that a lot size is in a range which is less than or equal to the range of from 5 to 10

acres;[8] the fourth that the last name is any string greater than the empty string (i.e., any nonempty string, given normal string-ordering conventions); and the fifth that for a feature whose value is a list of two strings, the first precedes the string ‘M' and the second is the string ‘M', or any string following it.

```
<f name="number.of.bedrooms"> <nbr value="5" rel="lt"/> </f>
<f name="illegal.speed"> <rate value="65" unit="miles" per="hour"
rel="gt"/> </f>
<f name="lot.size"> <msr value="5" valueTo="10" unit="acre" rel="le"/> </f>
<f name="last.name"> <str rel="gt"/> </f>
<f name="pairs" org="list"> <str rel="lt">M</str> <str rel="ge">M</str>
</f>
```

5.8.3 Subsumption and Non-subsumption Relations [16.9.3]

When the **`rel`** attribute is given the values `sb` or `ns`, the markup expresses the claim that the value given subsumes, or does not subsume, the actual value for the feature in question.
On the **`<str>`** element, these values are used to specify that the string value given in the **`<str>`** element is or is not a *substring* of the actual value of the feature.

Simons: Defining subsumption on string is something we did by analogy when developing the TEI spec. It is not a standard practice in the community. This would be a candidate for simplification.

The first example below specifies that the actual feature value may be any string at all (since the empty string is a substring of every string), the second that it might be any string in which the string ‘the' occurs as a substring, and the third that it might be any string in which the string ‘the' does not occur as a substring.

```
<str rel="sb"/>
<str rel="sb">the</str>
<str rel="ns">the</str>
```

On the **`<fs>`** element, the attribute values `sb` and `ns` indicate that the given feature structure does or does not legally *subsume* the actual feature structure. By definition, one feature structure subsumes another if the second feature structure is identical to the first or contains more information than the first. The default value for the **`rel`** attribute of the **`<fs>`** element is `sb`. The subsumption of feature structures is illustrated by the following four examples; suppose that the ‘person' and ‘number' features are either optional or obligatory in these `<fs type="agreement structure">` example elements.

```
<fs id="p3ns" type="agreement structure">
   <f name="person"> <sym value="third"/> </f>
   <f name="number"> <sym value="singular"/> </f>
</fs> <!-- third person singular -->
<fs id="p3nx" type="agreement structure">
   <f name="person"> <sym value="third"/> </f>
</fs> <!-- third person -->
<fs id="pxns" type="agreement structure">
   <f name="number"> <sym value="singular"/> </f>
</fs> <!-- singular -->
<fs id="pxnx" type="agreement structure"/> <!-- -->
```

The fourth example, **pxnx**, subsumes all four of the examples, since each contains at least as much information as does feature structure **pxnx**. Conversely, the first example, **p3ns**, subsumes only itself. Finally, the second and third examples, identified as **p3nx** and **pxns** attributes, subsume themselves and the first feature structure, but not each other.
If both person and number are obligatory features of agreement structure elements, then the last three

[8] We say that one range is less than or equal to another if both the value and valueTo attributes of the first are less than or equal to the corresponding attributes of the second.

elements in the preceding list have the same interpretation as their counterparts in the following list.

```
<fs id="p3na" type="agreement structure">
   <f name="person"> <sym value="third"/> </f>
   <f name="number"> <any/> </f>
</fs> <!-- third person -->
<fs id="pans" type="agreement structure" >
   <f name="person"> <any/> </f>
   <f name="number"> <sym value="singular"/> </f>
</fs> <!-- singular -->
<fs id="pana" type="agreement structure" >
   <f name="person"> <any/> </f>
   <f name="number"> <any/> </f>
</fs> <!-- -->
```

On the other hand, if both person and number are optional features of agreement structures, then those three elements have the same interpretation as their counterparts in the following list.

```
<fs id="p3nu" type="agreement structure">
   <f name="person"> <sym value="third"/> </f>
   <f name="number"> <uncertain/> </f>
</fs> <!-- 3d person -->
<fs id="puns" type="agreement structure">
   <f name="person"> <uncertain/> </f>
   <f name="number"> <sym value="singular"/> </f>
</fs> <!-- singular -->
<fs id="punu" type="agreement structure">
   <f name="person"> <uncertain/> </f>
   <f name="number"> <uncertain/> </f>
</fs> <!-- -->
```

That is, if an optional feature is omitted from a feature-structure representation, then that feature may have any of its legal values or the value **<uncertain>**.

The value `sb` is chosen as the default value for the **`rel`** attribute of the **<fs>** element, because it provides for the most economical means for underspecifying them. One situation in which it may be preferable to specify `<fs rel="eq">` is when the feature structure has many optional features and it is known that none of them occurs.

The specification `<fs rel="ns">` is used to denote the feature structures that the specified feature structure does not subsume. This provides a handy way of saying that a certain combination of features is not present, for example the combination of third person and singular number, as in the agreement structure of the English verb form ‘sink', understood as a present tense verb form. The following example expresses the claim that third-person and singular-number features are not both present in the agreement feature, but makes no further claim about what is present.

```
<f name="agreement">
   <fs id="np3ns" type="agreement structure" rel="ns">
      <f name="person"> <sym value="third"/> </f>
      <f name="number"> <sym value="singular"/> </f>
   </fs>
</f>
```

In most real situations, of course, one can infer, from the range of possible values for person and number, what the remaining possibilities are. Suppose, for example, that in the relevant feature system declaration, the features ‘person' and ‘number' are given the following **<vRange>** elements:

```
<vRange><!-- for the PERSON feature  -->
  <vAlt>
    <sym value='first'/>
    <sym value='second'/>
    <sym value='third'/>
  </vAlt>
</vRange>
<vRange><!-- for the NUMBER feature  -->
  <vAlt>
    <sym value='singular'/>
```

```
    <sym value='plural'/>
  </vAlt>
</vRange>
```
Suppose, further, that the person and number features are obligatory in feature structures of the type `agreement structure`. Then the element `<fs id="NP3NS">` above is equivalent to the following alternation; the features whose value is **`<any>`** may be omitted, since they are implied by the default value of `sb` for the **`rel`** attribute in the enclosing **`<fs>`** elements.
```
<vAlt id="p12na-panp">
   <fs id="p12na" type="agreement structure">
      <f name="person">
         <vAlt> <sym value="first"/> <sym value="second"/> </vAlt>
      </f>
      <f name="number"> <any/> </f>
   </fs>
   <fs id="panp" type="agreement structure">
      <f name="person"> <any/> </f>
      <f name="number"> <sym value="plural"/> </f>
   </fs>
</vAlt>
```
If, on the other hand, the person and number features were optional in feature structures of type `agreement structure`, then the interpretation of an underspecified feature structure will change. The element `<fs id="NP3NS">` given above is then equivalent to the following alternation; the features whose value is **`<uncertain>`** may be omitted as they are implied by the default subsumption relation holding between the structure given and the actual structure.
```
<vAlt id="p120nu-punp0">
   <fs id="p120nu" type="agreement structure">
      <f name="person">
         <vAlt> <sym value="first"/> <sym value="second"/> <none/> </vAlt>
      </f>
      <f name="number"> <uncertain/> </f>
   </fs>
   <fs id="punp0" type="agreement structure">
      <f name="person"> <uncertain/> </f>
      <f name="number">
         <vAlt> <sym value="plural"/> <none/> </vAlt>
      </f>
   </fs>
</vAlt>
```

5.8.4 Relations Holding with Sets, Bags, and Lists [16.9.4]

The **`rel`** attribute is also provided for the **`<f>`** element, but is designed to be used with that element only when its **`org`** attribute (see section 5.5 [16.6] Singleton, Set, Bag and List Collections of Values) is `set`, `bag`, or `list`. When associated with the **`<f>`** element, the **`rel`** attribute may take on any of the following four values: `eq`, `ne`, `sb`, and `ns`. The default value is `eq`. Consider first the use of the **`rel`** attribute with the **`<f>`** element when the given value of the feature is **`<null>`**.
```
<f name="extras" org="set"> <null/> </f>
<f name="extras" org="set" rel="ne"> <null/> </f>
<f name="extras" org="set" rel="sb"> <null/> </f>
<f name="extras" org="set" rel="ns"> <null/> </f>
```
The first example states that the ‘extras' feature has the null set as its value. The second example states that the ‘extras' feature is a set which is not equal to the null set. That is, its actual value might be any non-null set. The third example states that the ‘extras' feature has as its value a set of which the null set is a subset; that is to say, any set at all, including the null set. Note that this is not equivalent to the following, which states that the extras feature has as its value a single element which is any legal value for the ‘extras' feature, including for example a **`<str>`** element containing the value `pool`.
```
<f name="extras" org="set"> <any/> </f>
```

Finally, the fourth example states that the ‘extras' feature has as its value a set of which the null set is not a subset. Since the null set is a subset of every set, the fourth example in effect claims that the ‘extras' feature has no legal value; it is thus equivalent to the following, which states directly that the ‘extras' feature has no value.

```
<f name="extras" org="set"> <none/> </f>
```

Consider next the use of the **rel** attribute with the **<f>** element when the given value of the feature is a single **<str>** element with the content pool:

```
<f name="extras" org="set"> <str>pool</str> </f>
<f name="extras" org="set" rel="ne"> <str>pool</str> </f>
<f name="extras" org="set" rel="sb"> <str>pool</str> </f>
<f name="extras" org="set" rel="ns"> <str>pool</str> </f>
```

The first example states that the value of the ‘extras' feature is a set consisting of a single member, namely a **<str>** element containing the value pool. The second example states that the ‘extras' feature has as its value a set which is not equal to the set consisting of this particular member. It could, however, be a two-membered set, one of whose members is some other value. This example is thus not equivalent to the following, which states that the ‘extras' feature has as its value a set comprising a single member other than a **<str>** element with the content pool:

```
<f name="extras" org="set"> <str rel="ne">pool</str> </f>
```

The third example states that the ‘extras' feature has as its value any set of which the set consisting of the single member specified is a subset (i.e., any set which contains the element **<str>** with the value pool, and possibly others). Finally, the fourth example states that the ‘extras' feature has as its value any set which does not contain this element as a member.

5.8.5 Varieties of Subsumption and Non-subsumption [16.9.5]

The **rel** values sb and ns have different meanings depending on whether they occur within a **<str>**, **<fs>** or **<f>** element. However, the use of a common name for the value reflects a fundamental similarity in those meanings. For example, the value sb can be used in all three elements to indicate that the actual value is any string, any feature structure, or any set, bag or list, as follows. In the second example below, the **rel** attribute has not been specified, since it has the value sb by default on **<fs>** elements.

```
<str rel="sb"></str>
<fs></fs>
<f name="..." org="set" rel="sb"> <null/> </f>
<f name="..." org="bag" rel="sb"> <null/> </f>
<f name="..." org="list" rel="sb"> <null/> </f>
```

Because the value sb is not defined for the attribute **rel** on the **<nbr>**, **<msr>** and **<rate>** elements, the indication that a value may be any number, measure or rate is sometimes not quite as simple. Here is one way of specifying any positive or negative integer numeric value[9]

```
<vAlt>
   <nbr value="0" rel="gt" type="int"/>
   <nbr value="0" rel="le" type="int"/>
</vAlt>
```

The value ns also is understood in similar ways in the different elements in which it may occur. Above in this section, the equivalence of the following representations under certain conditions was shown (the **id** attributes and the redundant features with **<any/>** values have been omitted).

```
<f name="agreement">
   <fs type="agreement structure" rel="ns">
      <f name="person"> <sym value="third"/> </f>
      <f name="number"> <sym value="singular"/> </f>
   </fs>
</f>
```

[9] Typically, there will be no need to use an encoding like this one as the value of a feature, since the <any> element is available for that purpose. However, in setting up the feature declaration for that feature, it may be necessary to use such an encoding, precisely so as to provide an interpretation for the use of the <any> element as the value of that feature.

```
<f name="agreement">
   <vAlt>
      <fs type="agreement structure">
         <f name="person">
            <vAlt> <sym value="first"/> <sym value="second"/> </vAlt>
         </f>
      </fs>
      <fs type="agreement structure">
         <f name="number"> <sym value="plural"/> </f>
      </fs>
   </vAlt>
</f>
```

The value `ns` has an analogous meaning when the value in question is a set rather than a feature structure. Recast in such terms, the equivalence above still holds good:

```
<f name="agreement" org="set" rel="ns">
   <sym value="third"/>
   <sym value="singular"/>
</f>
<f name="agreement" org="set" rel="sb">
   <vAlt>
      <vAlt> <sym value="first"/> <sym value="second"/> </vAlt>
      <sym value="plural"/>
   </vAlt>
</f>
```

6 Bibliography

British National Corpus, http://www.hcu.ox.ac.uk/BNC/.

Carpenter, Bob (1992), *The Logic of Typed Feature Structures*, Cambridge University Press, Cambridge.

Copestake, Ann (2002), *Implementing Typed Feature Structure Grammars*, CSLI Publications, Stanford.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum, and Ivan Sag (1985), *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge, MA.

Ide, Nancy, Jacques Le Maitre, and Jean Veronis (1993), "Outline of a Model for Lexical Databases", *Information Processing and Management*, 29(2): 159-186. Reprinted in Antonio Zampolli et al. (eds.) (2001), *Current Issues in Computational Linguistics: In Honour of Don Walker*, Kluwer Academic Publishers, Dordrecht.

Johnson, Mark (1988), *Attribute-Value Logic and the Theory of Grammar*, CSLI Lecture Notes 16, Stanford.

Langendoen, D. Terence and Gary F. Simons (1995), "A rationale for the TEI recommendations for feature-structure markup", *Computers and the Humanities*, 29.

KATS: pages?

Pereira, Fernando C. N. (1987), *Grammars and Logics of Partial Information*, SRI International Technical Note 420, SRI International, Menlo Park, CA.

Sag, Ivan A. and Thomas Wasow (1999, 2003), *Syntactic Theory: A Formal Introduction*, CSLI Publications, Stanford.

Shieber, Stuart M. (1986), *An Introduction to Unification-Based Approaches to Grammar*, CSLI Lecture Notes 4, Stanford.

KATS: add

Kay, Martin (1992), "Unification", in Michael Rosner and Roderick Johnson (Eds.), *Computational Linguistics and Formal Semantics*, 1-30, Cambridge University Press.

Fenstad, Jens Erik, Tore Langholm, and Espen Vestre (1992), "Representations and interpretations", in Michael Rosner and Roderick Johnson (Eds.), *Computational Linguistics and Formal*

Semantics, 31-96, Cambridge University Press.

Simons: The TEI guideline chapters should be referenced as well.
Burnard : There are at least 2 references to the feature system declaration as section 6, but this is the bibliography (which, by the way, doesn't seem to reference TEI P4!)

Erjavec : I'd suggest adding at least the first HPSG book (Pollard & Sag 98) as it introduces typed FSs (although quite informally) and set the scene for the subsequent FS boom.

Pollard, Carl J. and Ivan A. Sag (1987), *Information-Based Syntax and Semantics*, CSLI Lecture Notes 13, Chicago University Press, Chicago.

KATS: add

Annex A (non-normative)

Examples for Illustration

Consider the problem of specifying the grammatical *case*, *gender* and *number* features of classical Greek noun forms. Assuming that the case feature can take on any of the five values `nominative`, `genitive`, `dative`, `accusative` and `vocative`; that the gender feature can take on any of the three values `feminine`, `masculine`, and `neuter`; and that the number feature can take on either of the values `singular` and `plural`, then the following may be used to represent the claim that the noun form ???? goddesses has accusative case, feminine gender and plural number.

```
<fs type="word structure">
   <f name="case">   <sym value="accusative"/> </f>
   <f name="gender"> <sym value="feminine"/> </f>
   <f name="number"> <sym value="plural"/> </f>
</fs>
```

An XML parser by itself cannot determine that particular values do or do not go with particular features; in particular, it cannot distinguish between the presumably legal encodings in the preceding two examples and the presumably illegal encoding in the following example.

```
<!-- *PRESUMABLY ILLEGAL* ... -->
<fs type="word structure">
   <f name="case">   <sym value="feminine"/> </f>
   <f name="gender"> <sym value="accusative"/> </f>
   <f name="number"> <minus/> </f>
</fs>
```

There are two ways of attempting to ensure that only legal combinations of feature names and values are used. First, if the total number of legal combinations is relatively small, one can simply list all of those combinations in **`<fLib>`** elements (together possibly with **`<fvLib>`** elements), and point to them using the **`feats`** attribute in the enclosing **`<fs>`** element. This method is suitable in the situation described above, since it requires specifying a total of only ten (5 + 3 + 2) combinations of features and values. Further, to ensure that the features are themselves combined legally into feature structures, one can put the legal feature structures inside **`<fsLib>`** elements. A total of 30 feature structures (5 3 2) is required to enumerate all the legal combinations of individual case, gender and number values in the preceding illustration. Of course, the legality of the markup requires that the **`feat`** attributes actually point at legally defined features, which an XML parser, by itself, cannot guarantee.

A more general method of attempting to ensure that only legal combinations of feature names and values are used is to provide a feature system declaration that includes a **`<valRange>`** element for each feature one uses. Here is a sample **`<valRange>`** element for the ‘case' feature described above; for further discussion of the **`<valRange>`** element, see chapter 6 [26] Feature System Declaration; the **`<vAlt>`** element is discussed in section 5.6 [16.7] Alternative Features and Feature Values.

[Note: <vAlt> may be replaced by a generic <alt> mechanism provided by the Linguistic Annotation Framework; e.g. <alt oper=”one”>…</alt>]

```
<!-- VALRANGE specification for CASE feature -->
```

Simons: But a generic <alt> mechanism could not constrain the kinds of values that can occur in particular contexts. For instance, in a <fs>, only <f>s are allowed, so an <fAlt> allows only <f>s in side of it. Whereas in an <f>, only feature values (and not <f>s) are allowed, so a <vAlt> only allows feature values.

The alternatives within feature structures are not generic alternatives. They are specific subtypes, namely, they are either alternations of features, or they are alternations of values.

Thus, a general <alt> seems like the wrong thing to do.

```
<valRange>
  <vAlt>
    <sym value='nominative'/>
    <sym value='genitive'/>
    <sym value='dative'/>
    <sym value='accusative'/>
    <sym value='vocative'/>
  </vAlt>
</valRange>
```

Similarly, to ensure that only legal combinations of features are used as the content of feature structures, one should provide **<fsConstraint>** elements for each of the types of feature structure one employs. For discussion of the **<fDecl>** and **<fsConstraint>** elements, see chapter 6 [26] Feature System Declaration. Validation of the feature structures used in a document based on the feature-system declaration, however, requires that there be an application program that can use the information contained in the feature-system declaration.

KATS: move this annex to sections 4.6 and 4.7

Annex B (informative)

Basic Operations on Feature Structures

Subsumption

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:

(1)

a. $\begin{bmatrix} word \\ \text{PHON } \text{'love'} \end{bmatrix}$

b. $\begin{bmatrix} word \\ \text{PHON: } \text{'love'} \\ \text{POS: noun} \end{bmatrix}$

The feature structure (a) says that the word is pronounced or spelled 'love' and that's all. But the feature structure (b) says more than that by providing the additional information that it is a noun. 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). Since it carries no information, the empty feature structure [ ] subsumes not only the feature structures (a) and (b), but also any other feature structures.

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. Formally speaking, the subsumption relation is a partial ordering over feature structures and is defined recursively as follows[10]:

Definition of Subsumption:

Given two feature structures, *A* and *B*, *A* is said to subsume *B*, written as A $\subseteq$ B in case that

i. **atomic case**: if they are both atomic, then *A* = *B*.
ii. **complex case**: if they are both complex, then the following conditions are satisfied:

A. For every path in *A*, the same path exists in *B* and its value in *A* subsumes its value in *B*,

B. for every pair of paths that is structure-sharing in A, the same pair of paths is structure-sharing in B, and

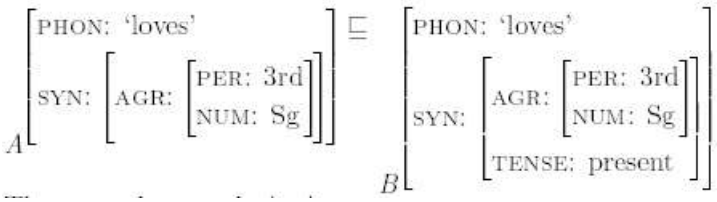
C. for every type assigned by A to a path subsumes the type assigned to the same path in B in the type ordering.

Clause (i) means that an atomic feature structure neither subsumes nor is subsumed by a different

[10] Carpenter (1992: 43) claims that the subsumption relation is a pre-ordering on the collection of feature structure "because it is possible to have two distinct feature structures that mutually subsume each other".

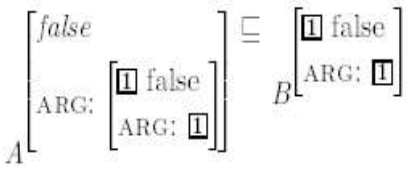
atomic feature structure (provided that it is not the empty feature structure). Each of the three conditions A, B, and C can be illustrated as below:

(2) Condition A on paths



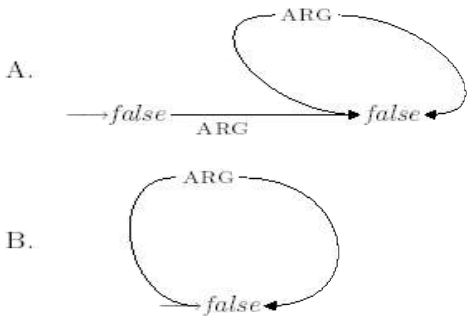
There are three paths in A: <PHON, SYN AGR PER, SYN AGR NUM>. These paths exist in B and each of their values are the same. Hence, A subsumes B by satisfying Condition A with the other two conditions being inapplicable[11].

(3) Condition B on structure sharing



The two feature structures above are cited in Carpenter (1992: 38, 39) as cases that support the cyclicity of feature structures. Here, A subsumes B because every path of shared structures in A is also found in B (Condition B), while satisfying the other two conditions. The following graphs show this relation more clearly.

(4) Cyclicity



Burnard : Talking of DAGs, I'm not sure that this mechanism can or should support cyclic graphs. There is a casual reference to these in footnote 3 which I think needs expansion, or removal.

KATS: DAGs are not cyclic by definition

(5) Condition C on type ordering

This condition applies only to typed feature structures under the assumption of some kind of type

[11] The tags A and B are attached to features structures for our present discussionsonly.

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:

(6) Case involving type ordering

$$A\begin{bmatrix} noun \\ \text{PHON: 'Mary'} \\ \text{AGR: } \begin{bmatrix} \text{PER: 3rd} \\ \text{NUM: Sg} \end{bmatrix} \end{bmatrix} \sqsubseteq B\begin{bmatrix} name \\ \text{PHON: 'Mary'} \\ \text{AGR: } \begin{bmatrix} \text{PER: 3rd} \\ \text{NUM: Sg} \\ \text{GENDER: feminine} \end{bmatrix} \end{bmatrix}$$

where $noun \sqsubseteq name$

The type of B is a subtype of the type A, thus A is considered as subsuming B. Furthermore, B has an extra piece of information about the gender. Hence, A properly subsumes B.

Unification

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

(7)

a. $$A\begin{bmatrix} noun \\ \text{AGR: } \begin{bmatrix} \text{PER: 3rd} \end{bmatrix} \end{bmatrix}$$

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

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

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, asshown below:

(8)

a. $$E\begin{bmatrix} noun \\ \text{AGR: } \begin{bmatrix} \text{PER: 3rd} \\ \text{NUM: Sg} \end{bmatrix} \end{bmatrix}$$

b. $$F\begin{bmatrix} verb \\ \text{AGR: } \begin{bmatrix} \text{PER: 3rd} \\ \text{NUM: Sg} \end{bmatrix} \end{bmatrix}$$

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.

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 $FS_1 \cup FS_2$. Compatible feature structures can be *unified* together to form a more (or at least equally) informative feature structure. The feature structure A, for instance, can be unified with C, yielding a little bit more enriched feature structure D.

(9) Unified feature structure

$$D\begin{bmatrix} noun \\ \text{AGR:} \begin{bmatrix} \text{NUM: Sg} \\ \text{PER: 3rd} \\ \text{GENDER: masculine} \end{bmatrix} \end{bmatrix}$$

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.

The operation of unification gets complicated when it involves shared structures. Consider the following example:

(10) Unification involving reentrancy

$$G\begin{bmatrix} word \\ \text{PHON: 'fish'} \\ \text{CAT: noun} \\ \text{AGR-CAT: } \boxed{1}\begin{bmatrix} agr \\ \text{PER: 3rd} \end{bmatrix} \\ \text{SUBJ: } \begin{bmatrix} word \\ \text{AGR-CAT: } \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup H\begin{bmatrix} phrase \\ \text{AGR-CAT: } \begin{bmatrix} agr \\ \text{NUM: plural} \\ \text{PER: 3rd} \end{bmatrix} \end{bmatrix}$$

$$= I\begin{bmatrix} word \\ \text{PHON: 'fish'} \\ \text{CAT: noun} \\ \text{AGR-CAT: } \boxed{1}\begin{bmatrix} agr \\ \text{PER: 3rd} \\ \text{NUM: plural} \end{bmatrix} \\ \text{SUBJ: } \begin{bmatrix} word \\ \text{AGR-CAT: } \boxed{1} \end{bmatrix} \end{bmatrix}$$

The unification of feature structures G and H resulted in a feature structure I. This unification involves structure sharing. Here, the value of AGR-CAT of H is unified with the value of the first

occurrence of AGR-CAT of G that is a feature structure tagged with the boxed integer 1. Furthermore, on the assumption that the type *word* is a lower type of the type *phrase*, the unified feature structure I is marked as of being the type *word*.

Annex C (normative)

Feature Structure DTD

Burnard : The DTD as presented here is incomplete: several of its elements are unreachable. I think a tag library style presentation might be more helpful.

Note: %om.RR;, %a.global (but that may be a mistake, since we still need 'id', at least); attributes and TEIform attributes removed from TEI original.

```
<!-- 16.2: Feature structures, binary values-->
<!ELEMENT fs ((f | fAlt | alt)*)>
```

Simons: Does <alt> mean something different than <fAlt> in this context?

```
<!ATTLIST fs
      %a.global;
      type CDATA #IMPLIED
      feats IDREFS #IMPLIED
      rel (eq|ne|sb|ns) "sb">
<!ELEMENT f ( null | ( plus | minus | any | none | dft | uncertain | sym |
nbr | msr | rate | str | vAlt | alt | fs )* )>
```

Simons: A parameter entity would be nice. This list appears many times in the DTD.

```
<!ATTLIST f
      %a.global;
      name NMTOKEN #REQUIRED
      org (single|set|bag|list) #IMPLIED
      rel (eq|ne|sb|ns) "eq"
      fVal IDREFS #IMPLIED>
<!ELEMENT plus EMPTY>
<!ATTLIST plus
      %a.global;>

<!-- end of 16.2-->

<!-- 16.3: Feature libraries-->
<!ELEMENT fLib ((f | fAlt)*)>
<!ATTLIST fLib
      %a.global;
      type CDATA #IMPLIED>
<!ELEMENT fsLib ((fs | vAlt)*)>
<!ATTLIST fsLib
      %a.global;
      type CDATA #IMPLIED>
<!ELEMENT fvLib ((plus | minus | any | none | dft | uncertain | null | sym
| nbr | msr | rate | str | vAlt)*)>
<!ATTLIST fvLib
      %a.global;
      type CDATA #IMPLIED>
<!-- end of 16.3-->
```

```
<!-- 16.4: Symbolic, etc. values-->
<!ELEMENT sym EMPTY>
<!ATTLIST sym
      %a.global;
      value CDATA #REQUIRED
      rel (eq|ne) "eq">
<!ELEMENT nbr EMPTY>
<!ATTLIST nbr
      %a.global;
      value CDATA #REQUIRED
      valueTo CDATA #IMPLIED
      rel (eq|ne|lt|le|gt|ge) "eq"
      type (int|real) #IMPLIED>
<!ELEMENT msr EMPTY>
<!ATTLIST msr
      %a.global;
      value CDATA #REQUIRED
      valueTo CDATA #IMPLIED
      unit CDATA #REQUIRED
      rel (eq|ne|lt|le|gt|ge) "eq"
      type (int|real) #IMPLIED>
<!ELEMENT rate EMPTY>
<!ATTLIST rate
      %a.global;
      value CDATA #REQUIRED
      valueTo CDATA #IMPLIED
      unit CDATA #IMPLIED
      per CDATA #REQUIRED
      rel (eq|ne|gt|ge|lt|le) "eq"
      type (int|real) #IMPLIED>
<!ELEMENT str (#PCDATA)>
<!ATTLIST str
      %a.global;
      rel (eq|ne|sb|ns|lt|le|gt|ge) "eq">
<!-- end of 16.4-->

<!-- 16.6: Null values-->
<!ELEMENT null EMPTY>
<!-- end of 16.6-->

<!-- 16.7: Alternative features and feature values-->
<!ELEMENT fAlt ((f | fs | fAlt), (f | fs | fAlt)+)>
<!ATTLIST fAlt
      %a.global;
      mutExcl (Y|N) #IMPLIED>
<!ELEMENT vAlt ((plus | minus | any | none | dft | uncertain | null | sym |
nbr | msr | rate | str | vAlt | fs), (plus | minus | any | none | dft |
uncertain | null | sym | nbr | msr | rate | str | vAlt | fs)+)>
<!ATTLIST vAlt
      %a.global;
      mutExcl (Y|N) #IMPLIED>
<!-- end of 16.7-->

<!-- 16.8: Boolean, default, uncertainty values-->
<!ELEMENT any EMPTY>
<!ELEMENT none EMPTY>
<!ELEMENT dft EMPTY>
<!ELEMENT uncertain EMPTY>
<!-- end of 16.8-->
```

Burnard : Further Comments:
There are at least two references to the linking mechanisms defined in P4. As these mechanisms are likely to be revised quite substantially at P5, I think it might be advisable to make some explicit statement about which of the various possible mechanisms is required by this standard. Some reference to xLink should also be included.

Lee Gillam: Further Comments:
From an implementation persepective, it makes sense to declare the various elements of feature structures and their operations (which I'd suggest forms a key part of the core rather than bringing up the rear in an annex). I also think it would work well to keep the notation away from the SGML/XML-based representation of the notation (which gives a cluttered feel at present). In keeping clean of the XML representation, which realistically is targeted at machine processing, although for some reason humans seem to like the effort of parsing it also, the feature structure standard would be able to provide a similar style/vocabulary approach already published in ISO 16642 (As of today, officially known in the UK as BS ISO 16642 - Hi Laurent). I'd go as far as to suggest putting the XML representation as an informative annex, where items refer to the relevant defining section, and perhaps operations and results can?be shown accordingly.