INTERNATIONAL STANDARD

ISO 24610-1

FIrst edition
2006-04-15

Language resource management — Feature structures —

Part 1: Feature structure representation

Gestion des ressources linguistiques — Structures de traits —

Partie 1: Représentation de structures de traits

Reference number
ISO 24610-1:2006(E)

© ISO 2006

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO 2006

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

--`,,```,,,,````-`-`,,`,,`,`,,`---

;;
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO 2006 – All rights reserved

Not for Resale

Contents

Page

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 24610-1 was prepared by Technical Committee ISO/TC 37, *Terminology and other language and content resources*, Subcommittee SC 4, *Language resource management*.

ISO 24610 consists of the following parts, under the general title *Language resource management — Feature structures*:

— *Part 1: Feature structure representation*

The following part is under preparation:

— *Part 2: Feature system declaration*

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

This part of ISO 24610 results from the agreement between the Text Encoding Initiative Consortium (TEI) and the ISO TC 37/SC 4 that a joint activity should take place to revise the two existing chapters on feature structures and feature system declaration in *The TEI Guidelines* called *P4.*

It is foreseen that ISO 24610 will have the following two parts.

— Part 1, *Feature structure representation,* describes feature structures and their representation. It provides an informal but explicit overview of their basic characteristics and formal semantics. In addition, part 1 defines a standard XML (eXtended Markup Language) vocabulary for the representation of untyped feature structures, feature values, and feature libraries. It thus provides a reference format for the exchange of feature structure representations between different application systems.

— Part 2, *Feature system declaration*, discusses ways of validating typed feature structures which are conformant to part 1, and of enforcing application-specific constraints. It proposes an XML vocabulary for the representation of such constraints with reference to a set of features and the range of values appropriate for them, and thus facilitates representation and validation of a type hierarchy as well as other well-formedness conditions for particular applications, in particular those related to the goal of language resource management.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Language resource management — Feature structures —

Part 1:
Feature structure representation

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 applications. This part of ISO 24610 provides a format for the representation, storage and exchange of feature structures in natural language applications concerned with the annotation, production or analysis of linguistic data. It also defines a computer format for the description of constraints that bear on a set of features, feature values, feature specifications and operations on feature structures, thus offering a means of checking the conformance of each feature structure with regards to a reference specification.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 8879, *Information processing — Text and office systems — Standard Generalized Markup Language (SGML)*, as extended by TC 2 (ISO/IEC JTC 1/SC 34 N029: 1998-12-06).

ISO 19757-2, *Information technology — Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*

NOTE The first reference permits the use of XML and the second, RELAX NG,provides a specification for XML modules. RELAX NG is a schema language for XML, standing for REgular LAnguage for XML for Next Generation, and simplifies and extends the features of DTDs, Document Type Definitions.

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 8879 and ISO 19757-2 and the following apply. This list is provided to clarify the terminology relating to feature structures used throughout this part of ISO 24610. Terminology derived from XLM and other formal languages is not defined here.

3.1
alternation
operation on feature **values** (3.23) that returns one and only one of the values supplied as its argument

NOTE Given a feature specification $F : a|b$, where $a|b$ denotes the alternation of a and b, F has either the value a or the value b, but not both.

3.2
atomic value
value (3.23) without internal structure, i.e. value other than **feature structure** (3.10) and **collection** (3.4)

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.3
boxed label
label in box used in a matrix notation to denote a value shared by several **features** (3.8)

NOTE The label may be any alphanumeric symbol.

3.4
collection
list, set, or multiset of **values** (3.23)

NOTE A list is an ordered collection of entities some of which may be identical. A set is an unordered collection of unique entities. A multiset is an unordered collection of entities that may or may not be unique; it is sometimes referred to as a bag.

3.5
complex value
value (3.23) represented either as a **feature structure** (3.10) or as **collection** (3.4)

3.6
concatenation
operation of combining two lists of **values** (3.23) into a single list

3.7
empty feature structure
feature structure (3.10) containing no **feature specifications** (3.9)

3.8
feature
property of an entity

NOTE The combination of feature and feature-value constitutes a **feature specification** (3.9). For example, number is a feature, singular is a value, and a pair <number, singular> is a feature specification.

3.9
feature specification
assignment of a **value** (3.23) to a **feature** (3.8)

NOTE Formally, it is treated as a pair of a feature and its value.

3.10
feature structure
set of **feature specifications** (3.9)

NOTE The minimum feature structure is the **empty feature structure** (3.7).

3.11
graph notation
notation of **feature structure** (3.10) in a single rooted graph

3.12
incompatibility
relation between two **feature structures** (3.10) which have conflicting **types** (3.19) or at least one common **feature** (3.8) with incompatible **values** (3.23)

NOTE Two feature structures that are incompatible cannot be unified. The **empty feature structure** (3.7) is compatible with any other feature structure.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.13
matrix notation
attribute-value matrix
AVM
notation that uses square brackets to represent **feature structures** (3.10)

NOTE In a matrix notation, each row represents a **feature specification** (3.9), with the feature name and the feature value separated by a colon (:), space ( ) or the equals sign (=).

3.14
merge
generic operation that includes **union** (3.22) of sets or multisets and **concatenation** (3.6) of lists

3.15
negation
(unary) operation on a **value** (3.23) denoting any other value incompatible with it

NOTE In this part of ISO 24610, negation applies to values only and is not understood as a truth function as in ordinary bivalent logics.

3.16
path
sequence of labeled arcs connecting nodes in a graph

3.17
structure sharing
re-entrancy
relation between two or more **features** (3.8) within a **feature structure** (3.10) that share a **value** (3.23)

3.18
subsumption
relationship between two **feature structures** (3.10) in which one is more specific than the other

NOTE A feature structure A is said to subsume a feature structure B if A is at least as informative as B. Subsumption is a reflexive, antisymmetric, and transitive relation between two feature structures.

3.19
type
name of a class of entities

NOTE **Feature structures** (3.10) may be characterized by grouping them into certain classes. Types are used to name such classes.

3.20
typed feature structure
feature structure (3.10) labelled by a **type** (3.19)

NOTE In the **graph notation** (3.11), each node is labelled with a type. In the **matrix notation** (3.13), a type is ordinarily placed at the upper left corner of the inside of the pair of square brackets that represents a typed feature structure. In XML notation, the type is supplied as the **value** (3.23) of a type attribute on the <fs> element.

3.21
unification
operation that combines two compatible **feature structures** (3.10) into the least informative feature structure that contains the information from the two

3.22
union
operation that combines two sets, or multisets, into one

NOTE The corresponding operation for lists is **concatenation** (3.6).

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.23
value
information about an entity

NOTE There are two kinds of feature values: **atomic value** (3.2) and **complex value** (3.5).

4 General characteristics of feature structure

4.1 Overview

A feature structure is a general purpose data structure that identifies and groups together individual features by assigning a particular value to each. 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 meta-language for representing analysis and interpretation of linguistic content. Moreover, this instantiation allows a specification of a set of features with values of specific types and restrictions, by means of feature system declarations, or other XML mechanisms discussed in ISO 24610-2[1)] .

4.2 Use of feature structures

Feature structures provide partial information about an object by specifying values for some or all of its features. For example, if a female employee named Sandy Jones who is 30 years old is of the present concern, then that person's sex, name and age can be specified in a succinct manner by assigning a value to each of these three features. These pieces of information can be put into a simple set notation, as in:

(1) Employee

{<SEX, *female*>, <NAME, *Sandy Jones*>, <AGE, *30*>}

Feature structures are generally used as a vehicle for linguistic descriptions. For example, the phoneme /p/ in English can be analysed in terms of its distinctive features: consonantal, anterior, voiceless, non-continuant or stop sound segment, etc. Each of these features may be combined with one or other of the binary values plus(+) and minus(-) to provide a feature specification. In a phonemic analysis such as the following, the value of a feature specifies the presence or absence of that feature:

(2) Sound segment /p/

{<CONSONANTAL, + >, <ANTERIOR, + >, <VOICED, ->, <CONTINUANT, ->}

In such an analysis, the sound segment /p/ is distinguished from other phonemes in terms of the presence or absence of specific features. For example, /p/ differs from the phoneme /b/ in VOICING, and from /k/ in articulatory position: one is articulated at the anterior, (the lip or alveolar area of the mouth), and the other at the nonanterior, namely the back of the oral cavity.

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

(3) Grammatical features of the verb "love"

{<POS, *verb*>, <VALENCE, *transitive*>, <SEMANTIC_RELATION, *loving*>},

where POS stands for part of speech.

1) Under preparation.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Since its first extensive use in generative phonology in mid-60s, the feature structure formalism has become an essential tool not only for phonology, but more generally in support of syntax and semantics as well as lexicon building, especially in computational work. Feature structures are used to describe and model linguistic entities and phenomena by specifying their properties. In the next clauses, some of the formal properties of feature structures are outlined together with means of representing them in a systematic manner.

4.3 Basic concepts

Feature structures may be viewed in a variety of ways. The most common and perhaps the most intuitive views are the following:

— a set of feature specifications that consists of pairs of features and their values;

— labelled directed graphs with a single root where each arc is labelled with the name of a feature and directed to its value.

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

(4) A feature structure as a set or partial function

$FS \subseteq \{<F_i, v_i> \mid F_i \in \mathbf{Feat}, v_i \in \mathbf{FeatVal}\}$

or

$FS : \mathbf{Feat} \longrightarrow \mathbf{FeatVal}$

where $\mathbf{FeatVal} = \mathbf{AtomVal} \cup \mathbf{FS}$ and where **FeatVal** stands for all possible values.

Values may be regarded as either atomic or complex. Atomic values are entities without internal structure, while complex values may be feature structures or collections of values (either complex or atomic).

NOTE These two definitions are not absolutely equivalent, for in a set there may be more than one value, v, for a feature, F.

For example, the part of speech (POS) feature can take the name of an atomic morpho-syntactic category such as verb as its value. Conversely, the agreement (AGR) feature in English takes a complex value in the form of a feature structure with features PERSON and NUMBER. The word “loves”, for instance, has a POS feature with the value “verb”, while the value of its AGR feature consists of a feature structure comprising two feature specifications: PERSON with the value “3rd”, and NUMBER with the value “singular”.

4.4 Notations

4.4.1 Overview

As a list of feature-value pairs, the overall form of a feature structure is simple. However, the internal structure of a feature structure may be complex when a feature structure contains either

a) a feature whose value is itself a feature structure, or

b) a multi-valued feature whose value is a list, set, or multiset.

Lists, sets and multisets may be made up of atomic values, or they may include complex values that are themselves feature structures. That is, feature structures allow limitless recursive embedding. It is therefore necessary to represent them in an understandable and mathematically precise notation.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

There are two commonly used notations for the representation of feature structures: 1) graphs; and 2) matrices. This part of ISO 24610 proposes a third notation based on the XML standard. The following subclauses discuss how the same feature structure may be represented using each of these equivalent notations. Graphs are suitable for mathematical discourses, matrices for linguistic descriptions, and XML notations for computational implementation.

4.4.2 Graph notation

Feature structures are often represented as labelled directed graphs with a single root.

NOTE This graph can be either

a) acyclic, in which case it is referred to as a directed acyclic graph (DAG), or

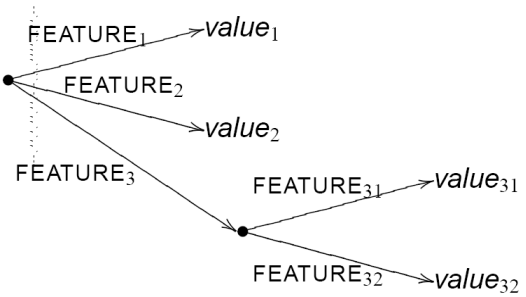
b) cyclic for handling cases like the Liar's paradox.

Feature structures are usually represented as DAGs, but it has been suggested that cyclical feature structures should be introduced to model some linguistic phenomena.

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

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

(5) Feature structure in graph notation



In this graph, the two features, FEATURE1 and FEATURE2, are atomic-valued, taking value1 and value2 on the terminal nodes respectively as their value. The feature FEATURE3 is, however, complex-valued, since it takes as its value the feature structure which is represented by the two arcs FEATURE31 and FEATURE32 with their respective values, value31 and value32.

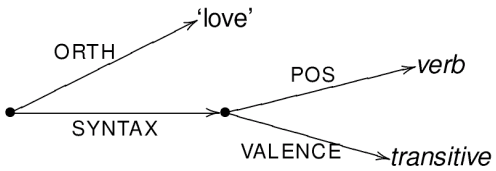
A graph may just consist of the root node, that is, without any branching arcs. Such a graph represents the empty feature structure.

There may be one or more arcs branching out from the root, each of them bearing a single feature name. Some of these labelled arcs originating from the root may again stretch out to another node and then from this node to another, forming an indefinitely long sequence of feature names. Such a sequence of feature names, labelling the arcs from the unique root node to each of the terminal nodes on a graph, is called a path. For example, there are four paths in (5): FEATURE1, FEATURE2, FEATURE3.FEATURE31 and FEATURE3.FEATURE32.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Here is a linguistically more relevant example.

(6) Linguistic example in graph notation



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

4.4.3 Matrix notation

Despite their mathematical elegance, graphs are difficult to typeset or read, especially when complex. For this reason, feature structures are more often depicted using a matrix notation called attribute-value matrix, or simply AVM.

NOTE 1 The term "attribute" is an alternative term for the concept which is called "feature".

(7) Matrix notation

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

Each row in the square bracket with a FEATURE name followed by its value name represents a feature specification in a feature structure.

NOTE 2 A colon, an equals sign or a space separates a feature from its value on each row of an AVM.

Feature values can be either atomic or complex. Each row with an atomic value terminates at that value. But if the value is complex, then that row leads to another feature structure, as in the case of FEATURE3 above.

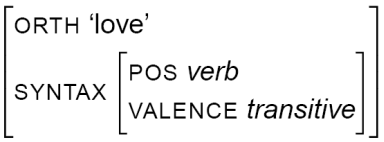
The notion of path is also important in several applications of the attribute-value matrix (AVM) notation, as discussed further below. A path in an AVM is a sequence of feature names, as is the case with feature structure graphs. An AVM with no rows and thus no occurrences of features, is represented as [ ]; this represents the empty feature structure. If an AVM has at least one row consisting of a feature name and its value, then there is a path of length 1 corresponding to each occurrence of a feature name in each row. Given a path of length i, if the last member of that path takes a nonempty feature structure as value, then that path forms a new path of length i +1 by taking each one of the features in that feature structure as its member.

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

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(8) Example of an AVM notation



This AVM has three paths: ORTH, SYNTAX.POS and SYNTAX.VALENCE.

4.4.4 XML-based notation

Like any other formalism, XML has its own terminology. An XML document consists of typed elements, occurrences of which are marked by start- and end-tags which enclose the content of the element. Every XML element may be regarded as a linearization of a tree with a single root node. Each node contains either terminal text or other element nodes. Elements have a specific type. Element occurrences can also carry named attributes (or, more exactly, attribute-value pairs). For example:

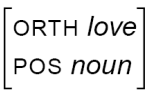
(9) <word class="noun">love</word>

This is the XML way of representing an occurrence of the element type word, the content of which is the string "love". The word element is defined as having an attribute called "class", whose value in this case is the string "noun".

The term "attribute" is thus used in a way which potentially clashes with its traditional usage in discussions of feature structures. In AVM, feature structures are represented in a square bracket form, with each row representing an attribute-value pair. In this usage, an attribute does not correspond necessarily to an XML attribute. To avoid this potential source of confusion, the reader should be aware that in the remainder of this clause, the term "attribute" will be used only in its technical XML sense.

To illustrate further the distinction, it should now be considered how a feature structure in AVM may be represented in XML. Consider the following AVM that represents a non-typed feature structure:

(10) Representation of a feature structure in AVM



This feature structure consists of two feature specifications: one combines the feature ORTH with the string value "love" and the other combines the feature POS with the string value "noun".

NOTE This representation says nothing about the range of possible values for the two features: in particular, it does not indicate that the range of possible values for the ORTH feature is not constrained, whereas (at least in principle) the range of possible values for the POS feature is likely to be constrained to one of a small set of valid codes. In a constraint-based grammar formalism, possible values for the feature ORTH need to be strings consisting of a finite number of characters drawn from a well-defined character set, say the Roman Alphabet plus some special characters, for each particular language.

In the XML representation proposed here, a feature structure is represented by means of an XML element fs, and a feature-value specification by means of an XML element f:

(11) <fs>
<f> ...</f>
<f> ...</f>

</fs>

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO 2006 – All rights reserved
Not for Resale

This is a more generic approach than another, equally plausible, representation, in which each feature specification might be represented by a specifically-named element:

(12)
```
<fs>
  <orth>...</orth>
  <pos>...</pos>
</fs>
```

Using this more generic approach means that systems can be developed which are independent of the particular feature set, at the expense of slightly complicating the representation in any particular case. By representing the specification of a feature as an f element, rather than regarding each specific feature as a different element type, the overall processing model is simplified considerably.

A feature specification, as has been seen, has two components: a name, and a value. Again, there are several equally valid ways of representing this combination in XML. Any of the following three, for example, could be chosen:

(13)

a)
```
<f name="pos" value="noun"/> ;
```

b)
```
<f name="pos">
    <value>noun</value>
</f> ;
```

c)
```
<f>
    <name>pos</name>
    <value>noun</value>
</f> .
```

The fact that all three of these are possible arises from a redundancy introduced in the design of the XML language largely for historical reasons. The present recommendation is to use a formulation like b) above, though c) may be preferable on the grounds of its greater simplicity.

Finally, the representation of the value part of a feature specification is discussed. A name is simply a name, but a value in the system discussed here may be of many different types: it might for example be an arbitrary string, one of a predefined set of codes, a Boolean value, or a reference to another (nested) feature structure. In the interests of greater expressivity, the present system proposes to distinguish amongst these kinds of value in the XML representation itself.

Once more, there are a number of more or less equivalent ways of doing this. For example:

(14)

a)
```
<fs>
    <f name="orth">
    <value type="string">love</value></f>
   <f name="pos">
   <value type="symbol">noun</value></f>
</fs> ;
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

b) <fs>

<f name="orth"><string>love</string></f>

<f name="pos"><symbol value="noun"/></f>

</fs> .

The number of possible different value types is comparatively small, and the advantages of handling values generically rather than specifically seem less persuasive. The approach currently taken is therefore to define different element types for the different kinds of value (for example, string to enclose a string, symbol to denote a symbol, binary to denote a binary value, fs to denote a nested feature structure).

Representing feature structures in XML notation is thus possible and rather straightforward. For example, the following is the XML representation for the feature structure given in (6) and (8) above.

(15) Feature structure in XML notation

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

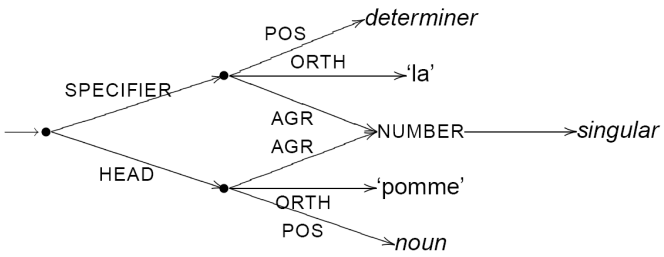
Despite the apparent difference between the representations (6), (8) and (15), the information contained in each is identical. For example, the arc labels in graph notation like ORTH and SYNTAX are now represented by <f> tags with appropriate names. Furthermore, the XML representation can be processed by general purpose software tools and thus can be converted automatically to a variety of other formats.

NOTE A more detailed discussion including ways to simplify the representation in (15) is provided in Clause 5.

4.5 Structure sharing

A feature structure representation may need to represent shared components. For example, in representing the structure of a noun phrase (“la pomme") which includes a determiner (article) “la”, it may be desired to show that the feature NUMBER is shared between the determiner and the noun. This sharing can be directly represented in the graphic notation as follows:

(16) Merging paths in graph notation



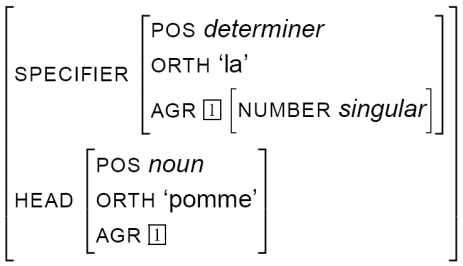
where POS again stands for part of speech, ORTH for orthography and AGR for agreement feature.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

In AVM notation, structure sharing is represented by providing a boxed integer label at the point where structures are shared, as in the following example:

(17) Structure sharing in AVM notation



In a similar way, in the XML notation the element *var* is used to provide a name for the sharing point. It supplies a label for the point which can then be referenced elsewhere in the structure, as in the following example:

(18) Structure sharing in XML notation

```
<fs>
  <f name="specifier">
    <fs>
      <f name="agr">
          <vLabel name="n1">
            <fs>
            <f name="number"><symbol value="singular"/></f>
            </fs>
          </vLabel>
      </f>
      <f name="pos"><symbol value="determiner"/></f>
    </fs>
  </f>
  <f name="head">
    <fs>
      <f name="agr"><vLabel name="n1"/></f>
      <f name="pos"><symbol value="noun"/></f>
    </fs>
  </f>
</fs>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Re-entrancy is symmetric and so there is no reason to distinguish amongst the various occurrences of a shared node. In particular, this implies that a value may be attached to any or all occurrences of a shared label:

(19) Specifying shared values

```
<fs>
 <f name="specifier">
  <fs>
   <f name="agr"><vLabel name="n1">
    <fs>
     <f name="number"><symbol value="singular"/></f>
    </fs></vLabel></f>
   <f name="pos"><symbol value="determiner"/></f>
  </fs>
 </f>
 <f name="head">
  <fs>
   <f name="agr"><vLabel name="n1">
    <fs>
     <f name="number"><symbol value="singular"/></f>
    </fs></vLabel></f>
   <f name="pos"><symbol value="noun"/></f>
  </fs>
 </f>
</fs>
```

A particularly significant case of structure sharing arises when the aim is to express that two features have the same value, even though that value is not known or is underspecified.

4.6 Collections as complex feature values

4.6.1 Overview

In feature-based grammar formalisms, such as Head-driven Phrase Structure Grammar (HPSG) and Lexical Functional Grammar (LFG), multi-valued features, taking collections as their values are very common. Collections may be organized as lists, sets, or multisets (also known as bags). The items in a list are ordered, and need not be distinct. The items in a set are not ordered, and need to be distinct. The items in a bag are neither ordered nor distinct. 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.

Collections of any kind can be represented directly in both AVM and XML notations. There is however no theoretical consensus on how they should be represented using graph notation.

4.6.2 Lists as feature values

A well-known example of a list-valued feature is the feature SUBCAT, which stands for subcategory in Head-driven Phrase Structure Grammar. It is used to describe the kind of a grammatical subject and objects that a verb expects (“subcategorizes for”). For example, to represent that the English verb form “gives”, as used in structures like “John gives Mary a kiss”, expects a nominative noun phrase as the subject, a dative noun

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

phrase as its indirect object and an accusative noun phrase as the direct object, and expects these elements in this order, the feature SUBCAT is given the value:

(20) SUBCAT as a list-valued feature

$$\left[\text{SUBCAT}\left\langle\begin{bmatrix}\text{POS } \textit{noun}\\ \text{CASE } \textit{nominative}\end{bmatrix},\begin{bmatrix}\text{POS } \textit{noun}\\ \text{CASE } \textit{dative}\end{bmatrix},\begin{bmatrix}\text{POS } \textit{noun}\\ \text{CASE } \textit{accusative}\end{bmatrix}\right\rangle\right]$$

where the pair of angled brackets is used to represent a list as in set theory.

The AVM representation (20) can be represented in XML by the element **vcoll** which carries an attribute **org** whose value is a list, as below.

(21) Representing the value of SUBCAT as a list

```
<fs>
 <f name="subcat">
  <vColl org="list">
   <fs>
    <f name="pos"><symbol value="noun"/></f>
    <f name="case"><symbol value="nominative"/></f>
   </fs>
   <fs>
    <f name="pos"><symbol value="noun"/></f>
    <f name="case"><symbol value="dative"/></f>
   </fs>
   <fs>
    <f name="pos"><symbol value="noun"/></f>
    <f name="case"><symbol value="accusative"/></f>
   </fs>
  </vColl>
 </f>
</fs>
```

When treated as a feature value, a list may contain either atomic or complex values, as shown above. List values can also be represented recursively, as in this example:

(22) Recursive representation

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

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

or, equivalently in XML:

(23) Recursive representation of a list in XML

```
<fs>
 <f name="F">
  <fs>
   <f name="first"><symbol value="a"/></f>
   <f name="rest">
    <fs>
     <f name="first"><symbol value="b"/></f>
     <f name="rest"><symbol value="null"/></f>
    </fs>
   </f>
  </fs>
 </f>
 <f name="G">
  <fs>
   <f name="first">
    <fs>
     <f name="A"><symbol value="a"/></f>
    </fs>
   <f name="rest">
    <fs>
     <f name="first">
      <fs>
       <f name="B"><symbol value="b"/></f>
      </fs>
     <f name="rest"><symbol value="null"/></f>
    </fs>
   </f>
  </fs>
 </f>
</fs>
```

4.6.3 Sets as feature values

Features taking sets as their values are frequently used to represent grammatical features of languages like Japanese, Korean or German, in which word order is free or semi-free. In such languages, the subject and the verbal complements in a sentence have no fixed order.

NOTE There has been some controversy among linguists concerning the validity of presenting semi-free or free word order as supporting evidence for the introduction of sets as feature values.

Hence, for these languages, the feature SUBCAT (subcategory) or COMPS (complements), as well as the feature ARG_ST (argument structure) may be treated as taking a set, not a list of complements or arguments as its value. For the German equivalent of “gives”, the verb form “gibt”, the following would hold:

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(24) Set-valued features

$$\begin{bmatrix} \text{ORTH } \text{‘gibt’} \\ \text{POS } \textit{verb} \\ \text{SUBCAT } \left\{ \begin{bmatrix} \text{POS } \textit{noun} \\ \text{CASE } \textit{nominative} \end{bmatrix}, \begin{bmatrix} \text{POS } \textit{noun} \\ \text{CASE } \textit{dative} \end{bmatrix}, \begin{bmatrix} \text{POS } \textit{noun} \\ \text{CASE } \textit{accusative} \end{bmatrix}, \right\} \end{bmatrix}$$

The AVM representation in (24) can be represented in XML in an equivalent manner.

(25) Set-valued features in XML

```
<fs>
  <f name="orth"><string>gibt</string></f>
  <f name="pos"><symbol value="verb/></f>
  <f name="subcat">
   <vColl org="set">
          <fs>
     <f name="pos"><symbol value="noun"/></f>
     <f name="case"><symbol value="nominative"/></f>
    </fs>
    <fs>
     <f name="pos"><symbol value="noun"/></f>
     <f name="case"><symbol value="dative"/></f>
    </fs>
    <fs>
     <f name="pos"><symbol value="noun"/></f>
     <f name="case"><symbol value="accusative"/></f>
    </fs>
   </vColl>
  </f>
</fs>
```

Unlike lists, sets may not be defined recursively.

4.6.4 Multisets as feature values

For completeness, the possibility of multiset-valued features is included, although there are few applications for them. The set-valued feature SLASH in Head-driven Phrase Structure Grammar, for instance, is used for dealing with *wh*-movement and other extraction-like phenomena, where the values of SLASH contain the properties of the extracted gaps and these gaps can at times be identical.

For illustration, consider the following example:

(26) Coreferential pronouns

$\text{John}_{1=2}$ is an idiot. But he_1 thinks he_2 is smart.

The two occurrences of the pronouns above are coreferential. A multiset value might be used to represent this coreference as follows:

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(27) Coreferential multiset

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

An XML representation equivalent to (27) would be:

(28) Representing coreferential multiset in XML

```
<vColl org="multiset">
 <fs>
  <f name="pos"><symbol value="pronoun"/></f>
  <f name="person"><symbol value="3rd"/></f>
  <f name="number"><symbol value="singular"/></f>
  <f name="gender"><symbol value="masculine"/></f>
 </fs>
 <fs>
  <f name="pos"><symbol value="pronoun"/></f>
  <f name="person"><symbol value="3rd"/></f>
  <f name="number"><symbol value="singular"/></f>
  <f name="gender"><symbol value="masculine"/></f>
 </fs>
</vColl>
```

Unlike lists, multisets cannot be defined recursively.

4.7 Typed feature structure

4.7.1 Overview

In many recent grammar formalisms, the typed feature structure has become a major component of linguistic description.

4.7.2 Types

Elements of a domain can be sorted into classes called types in a systematic way, based on the presence of common properties amongst them. Usually, one or other of the features of a non-typed feature structure can be identified as a means of grouping commonly co-occurring features, which may thus be regarded as types. Linguistic entities such as phrase, word, POS (part of speech), noun, and verb which might be treated as features of a non-typed feature structure, could be taken as a type in a typed feature structure.

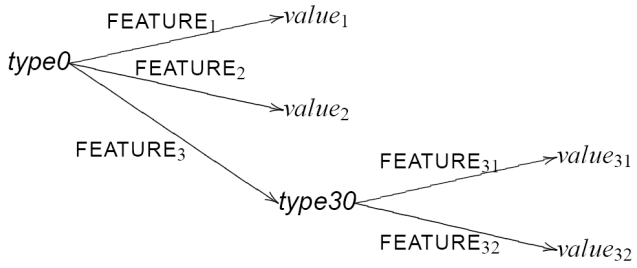
By typing it, it becomes possible to constrain the content of a feature structure. A feature structure of the type “noun”, for instance, would not allow a feature like TENSE in it or a specification of its feature CASE with a value such as feminine.

4.7.3 Notations

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

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(29) Typed feature structure in graph notation



The only difference between the typed graph (29) and the non-typed graph (5) is that each of the two nodes has been assigned a type: one is of type0 and the other of type30.

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

(30) Typed feature structure in AVM notation

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

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

NOTE Instead of the previous feature names like word, noun and verb, new names like word_st, noun_st and verb_st are used as type names.

The same information can be represented in an XML notation, as below:

(31) Typed feature structure in XML notation

```
<fs type="word_st">
   <f name="orth"><string>love</string></f>
   <f name="syntax">
      <fs type="verb_st">
         <f name="valence"><symbol value="transitive"/></f>
      </fs>
   </f>
</fs>
```

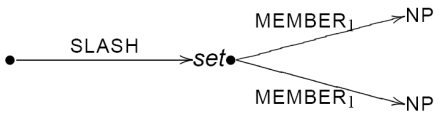
The XML attribute *type* may be used to specify a type for the <fs> element, rather than simply specifying the type as an embedded feature within the feature structure. Thus, in this example, the feature <f name="pos"><sym value="verb"/></f> in the embedded feature structure <fs> has been replaced by the typed feature structure <fs type="verb_st">.

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

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(32) Set-valued feature SLASH in graph notation



Here, the features MEMBER1 and MEMBER2 should be considered appropriate for the type “set”.

4.8 Subsumption: relation on feature structures

4.8.1 Overview

One important aspect of the use of feature structures is their ability to capture and represent partial information. No feature structure is expected to represent the totality of information describing all possible worlds or states of affairs. Instead, a feature structure analysis focuses on those particular aspects which are of interest or value in a particular situation, and captures only a subset of the possibilities. The notion of subsumption is used to specify which of a group of feature structures relating to the same information carries more information than the others.

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

(33)

a)
$$\begin{bmatrix} \textit{word\_st} \\ \text{ORTH } \text{‘loves’} \end{bmatrix}$$

b)
$$\begin{bmatrix} \textit{word\_st} \\ \text{ORTH } \text{‘loves’} \\ \text{SYNTAX} \begin{bmatrix} \textit{verb\_st} \\ \text{FORM } \textit{finite} \end{bmatrix} \end{bmatrix}$$

The feature structure a) says that the word is a string consisting of 5 letters, spelled as “loves”, and that is all. But the feature structure b) says more than that, by providing the additional information that it is a finite verb. Hence, a) is said to be less informative than b).

The technical term “subsumption” is used to describe this relation amongst feature structures; in the above case, a) is said to subsume b).

4.8.2 Definition

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

NOTE Carpenter (1992, p. 43)[7] claims that the subsumption relation is a preordering on the collection of feature structures. It is transitive and reflexive, but not antisymmetric because it is possible to have two distinct feature structures that mutually subsume each other. But these are alphabetic variants.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(34) Definition of subsumption

Given two typed feature structures, *FS*1 and *FS*2, *FS*1 is said to subsume *FS*2, written as *FS1 (angled subset relation symbol) FS2* if and only if the following conditions hold.

— **A. Path values:** Every path **P** which exists in *FS*1 with a value of type **t** also exists in *FS*2 with a value which is either **t** or one of its subtypes.

— **B. Path equivalences:** Every two paths which are shared in *FS*1 are also shared in *FS*2.

— **C. Type ordering:** Every type assigned by *FS*1 to a path subsumes the type assigned to the same path in *FS*2 in the type ordering.

Each of the three conditions A, B, and C can be illustrated (see 4.8.3, 4.8.4 and 4.8.5).

4.8.3 Condition A on path values

(35) Example satisfying Condition A

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

where verb_st again stands for verb structure, AGR for agreement, and agr-cat for agreement category.

There is only one path in *A*: <AGR.NUMBER>. This path exists in *B* and their values are the same.

NOTE The labels *A* and *B* in the lower left corner of each AVM are provided for convenience when reading this discussion only; they do not form part of the feature structure being represented.

Hence, Condition A is satisfied. Condition B is inapplicable here, since there is no structure sharing in either of the two feature structures. Condition C is satisfied because every type assigned by *A* to a path is identical with the type assigned to the same path in *B*. Hence, *A* subsumes *B*.

4.8.4 Condition B on structure sharing

(36) Case satisfying Condition B

$$A\begin{bmatrix} \textit{syn-cat} \\ \text{VALENCE} \begin{bmatrix} \textit{val-cat} \\ \text{SPECIFIER} \langle \boxed{1} \text{ NP} \rangle \\ \text{COMPS} \langle\rangle \end{bmatrix} \\ \text{ARG-ST} \langle \boxed{1} \rangle \end{bmatrix} \sqsubseteq B\begin{bmatrix} \textit{syn-cat} \\ \text{ORTH } \text{‘walks’} \\ \text{HEAD} \begin{bmatrix} \textit{verb\_st} \\ \text{FORM finite} \end{bmatrix} \\ \text{VALENCE} \begin{bmatrix} \textit{val-cat} \\ \text{SPECIFIER} \langle \boxed{1} \text{ NP} \rangle \\ \text{COMPS} \langle\rangle \end{bmatrix} \\ \text{ARG-ST} \langle \boxed{1} \rangle \end{bmatrix}$$

where syn-cat stands for syntactic category, val-cat valence category, COMPS complements, ORTH orthography or spelling, verb_st verb structure, NP noun phrase, and ARG-ST argument structure.

This example looks a little complicated. But one can easily check that the structure sharing tagged by $\boxed{1}$ in *A* also exists in *B* and the other two conditions are also satisfied. Hence, the subsumption relation holds here. Consider the following pair of examples related to the structure sharing condition:

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(37) Another case involving structure sharing

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

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

where word_st stands for word structure, SYN syntax, syn-cat syntactic category, noun_st noun structure, AGR agreement, agr-cat agreement category and val-cat valence category.

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

4.8.5 Condition C on type ordering

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

(38) Case involving type ordering

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

where $\textit{noun} \sqsubseteq \textit{name}$

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

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.9 Operations on feature structures and feature values

4.9.1 Overview

Two further operations may be defined over feature structures: unification and generalization. The former augments the amount of information represented in one feature structure by combining it with another; by contrast, the latter selects identical feature specifications and puts them into one general feature structure. In addition to these operations on feature structures, operations may be defined over feature values or types. Examples discussed in this section include the concatenation (symbolized by a circled plus) of list values and the alternation of feature values and conjunctive types.

4.9.2 Compatibility

Some feature structures are compatible with some others, while there are conflicting cases. Consider the following three cases, represented in AVM notation.

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

(39)

a. $A\left[\begin{array}{l} \textit{noun\_st} \\ \text{AGR}\ \left[\text{PERSON}\ \textit{3rd}\right] \end{array}\right]$

b. $B\left[\begin{array}{l} \textit{noun\_st} \\ \text{AGR}\ \left[\begin{array}{l} \text{NUMBER}\ \textit{singular} \\ \text{GENDER}\ \textit{feminine} \end{array}\right] \end{array}\right]$

c. $C\left[\begin{array}{l} \textit{noun\_st} \\ \text{AGR}\ \left[\begin{array}{l} \text{PERSON}\ \textit{3rd} \\ \text{GENDER}\ \textit{masculine} \end{array}\right] \end{array}\right]$

The feature structure *A* is said to be compatible with *B* and also with *C*. But the feature structures *B* and *C* are incompatible because their specification of the feature GENDER is conflicting: one has the value “feminine” and the other the value “masculine”.

Incompatibility may also arise when there is a type difference, as shown below:

(40) Incompatible feature structures

a. $E\left[\begin{array}{l} \textit{noun\_st} \\ \text{AGR}\ \left[\begin{array}{l} \text{PERSON}\ \textit{3rd} \\ \text{NUMBER singular} \end{array}\right] \end{array}\right]$

b. $F\left[\begin{array}{l} \textit{verb\_st} \\ \text{AGR}\ \left[\begin{array}{l} \text{PERSON}\ \textit{3rd} \\ \text{NUMBER}\ \textit{singular} \end{array}\right] \end{array}\right]$

The feature structures *E* and *F* both have the same agreement feature specifications, but these two feature structures are incompatible because they belong to two different types which are not in a subsumption relation.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.9.3 Unification

Compatible feature structures represent different aspects of information concerning a single entity. It is thus often desirable to merge them together to provide a coherent view. This merging process is captured by the operational process of unifying two compatible feature structures, *FS*1 and *FS*2, where the unification is represented by *angled U*. Compatible feature structures can be unified together to form a more (or at least equally) informative feature structure. The unification of two typed feature structures *FS*1 and *FS*2 is the most informative typed feature structure which is subsumed by both *FS*1 and *FS*2, if it exists.

NOTE 1 Formally speaking, the unification of two incompatible feature structures results in inconsistency and may just be represented with some inconsistency symbol like **T** without calling into the procedural aspect of its failure. With this symbol, unification may be treated as a total operation that always yields a result even with the inconsistency **T**.

(41) Formal definition of unification

> The unification of two typed feature structures *FS1* and *FS2* is the least upper bound of *FS1* and *FS2* in the collection of typed feature structures ordered by subsumption.

It should be noted here that the type hierarchy is constructed with the most non-specific or the least informative type at the bottom, which is represented as $\perp$.

NOTE 2 If the type hierarchy is depicted with the most general or non-specific type at the top **T**, then the unification of two typed feature structures *FS*1 and *FS2* needs to be defined as the greatest lower bound of *FS1* and *FS2* in the type hierarchy and also needs to be represented as *FS1* $\sqcap$ *FS2,* as in Copestake (2002)[12].

The feature structure *E* (=40a), for instance, can be unified with *C* (=39c), yielding a slightly more enriched feature structure *D* (=42).

(42) Unified feature structure

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

As shown here, unification normally adds information. However, identical features may also be unified without adding any further information. The empty feature structure may be unified with every feature structure without changing the content of the latter, and is thus formally treated as the identity element of unification.

(43) Basic properties of unification

- Unification adds information.
- Unification is idempotent: the unification of identical feature structures remains the same without anything added.
- The empty structure is the identity element for unification: it adds nothing to the resulting feature structure.

Unification may thus be considered an analogue of the set-theoretic union. Hence, they are usually represented by the same angled U symbol.

4.9.4 Unification of shared structures

The operation of unification becomes complicated when shared structures are involved. Consider the following example.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(44) Unification involving re-entrancy

$$G\begin{bmatrix} verb\_st \\ \text{AGR } \boxed{1} \\ \text{SPECIFIER} < \begin{bmatrix} noun\_st \\ \text{AGR} \boxed{1} \end{bmatrix} > \end{bmatrix} \sqcup H\begin{bmatrix} verb\_st \\ \text{AGR } \begin{bmatrix} \text{PERSON } 3rd \end{bmatrix} \end{bmatrix}$$

$$= I\begin{bmatrix} verb\_st \\ \text{AGR } \boxed{1}\begin{bmatrix} \text{PERSON } 3rd \end{bmatrix} \\ \text{SPECIFIER} < \begin{bmatrix} noun\_st \\ \text{AGR} \boxed{1} \end{bmatrix} > \end{bmatrix}$$

The unification of feature structures *G* and *H* results in a feature structure *I*. This unification involves structure sharing. Here, the AGR value of *H* in the first path is simply unified with the AGR value of *G* occurring in the first path which is tagged with $\boxed{1}$. The resulting feature structure *I* becomes of type verb_st.

On the assumption that the type verb_st is a subtype of the type agr-pos (agreement part of speech), consider a slightly more complicated case:

(45) Case for pumping values to each other

$$G\begin{bmatrix} verb\_st \\ \text{AGR } \boxed{1} \\ \text{SPECIFIER} < \begin{bmatrix} noun\_st \\ \text{AGR} \boxed{1} \end{bmatrix} > \end{bmatrix} \sqcup J\begin{bmatrix} agr\text{-}pos \\ \text{AGR } \begin{bmatrix} \text{PERSON } 3rd \end{bmatrix} \\ \text{SPECIFIER} < \begin{bmatrix} noun\_st \\ \text{AGR } \begin{bmatrix} \text{NUMBER } singular \end{bmatrix} \end{bmatrix} > \end{bmatrix}$$

$$= K\begin{bmatrix} verb\_st \\ \text{AGR } \boxed{1}\begin{bmatrix} \text{PERSON } 3rd \\ \text{NUMBER } singular \end{bmatrix} \\ \text{SPECIFIER} < \begin{bmatrix} noun\_st \\ \text{AGR} \boxed{1} \end{bmatrix} > \end{bmatrix}$$

As before, the two occurrences of the path AGR in *G* and *J* are unified, yielding the first path as in *K*. Another pair of the two occurrences of the path SPECIFIER.AGR are also unified. As a result, this path in *K* shares the value with the path AGR as is marked with $\boxed{1}$ and then its original value can be transferred to the place where the index $\boxed{1}$ first has occurred, namely in the path AGR.

4.10 Operations on feature values and types

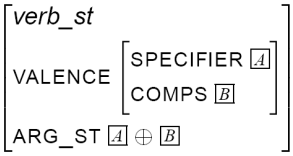
4.10.1 Concatenation and union operations

A list as a feature value may be appended to another list by *concatenation,* symbolized by a circled plus. In Head-driven Phrase Structure Grammar, the value of argument structure ARG_ST is treated as the concatenation of the SPECIFIER and COMPS value lists.

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(46) Concatenated list



Here, variable lists are tagged with boxed upper case Latin characters. Hence, the value of ARG_ST (argument structure) should be understood as the concatenation of two lists.

The similar operation of combining the members of two sets or multisets is known as union. These operations are theoretically possible, but are rarely used in linguistic description.

In the XML notation, the **vcoll** element is used to represent both concatenation and union. Thus, Example (46) can be represented in XML as follows:

(47) Representing a list of values in XML

```
<fs type="verb_st">
  <f name="valence">
    <fs>
      <f name="specifier">
        <var label="nA"></var>
      </f>
      <f name="comps">
        <var label="nB"></var>
      </f>
    </fs>
  </f>
  <f name="arg_st">
    <vColl org="list">
      <var label="nA"></var>
      <var label="nB"></var>
    </vColl>
  </f>
</fs>
```

4.10.2 Alternation

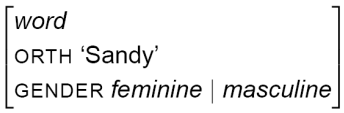
Some features may take an alternative value.

NOTE Strictly speaking, this alternation differs from the Boolean disjunction, or the inclusive disjunction, often represented by a wedge V. Disjunction results in truth if and only if one of its disjuncts is true. Alternation, however, allows the choice of only one value. Hence, the mathematical characterization of feature structure as a function still remains valid. The same is true with those cases in which features have collections like lists, sets or multisets as values, since they are not allowed to have more than one multivalue as their values.

For example, the word “Sandy” can be either feminine or masculine, but not neuter. Hence, the value of its gender can be specified with the vertical bar “|” as follows.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(48) Alternative value



Many cases that require an alternative feature value can be handled just by underspecifying or omitting feature specifications. But in the above example, it is not possible. If there were only two possible values for the gender in English, one could have just skipped specifying its value. But since there is a third value for gender which needs to be excluded, namely "neuter", one needs to specify the value as an alternative.

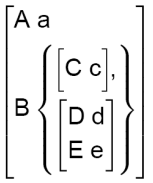
Alternation may be represented in the XML notation by means of the <**vAlt**> element:

(49) Representing alternative value in XML

```
<fs type="word">
  <f name="orth">
  <string>Sandy</string>
  </f>
  <f name="gender">
    <vAlt>
    <symbol value="feminine"></symbol>
    <symbol value="masculine"></symbol>
    </vAlt>
  </f>
</fs>
```

Curly brackets "{ }" are sometimes used to represent the alternation of feature values, especially when it is in the form of a full-blown feature structure. The following is an abstract and yet simple example:

(50) Alternative feature structures as feature value



Here the value of the feature B is specified with an alternative value, which consists of two feature structures. However, since curly brackets are also used to represent set feature values, their usage for alternation should be avoided to reduce the possibility of confusion.

4.10.3 Negation

Negation may be regarded as either truth-functional or as set-theoretic. In the former case, as in Boolean bivalent logic, negation is a truth function that returns the opposite value to a value: it assigns truth to falsity and vice versa. In set-theoretic terms, on the other hand, the negation of a value which is a member of some set *A* is understood as being some value in the complement *A'* of that value set.

In this part of ISO 24610, negation applies only to the value of a feature in a feature structure and is understood in a set-theoretic sense. If the value of a feature is atomic, then its negation is a member of the complement of the value set to which that value belongs. If the value set has only two members, then negation returns whichever of the two was not given as its argument. For example, if the feature number is

© ISO 2006 – All rights reserved

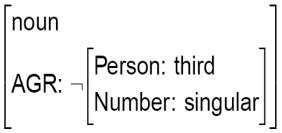
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

singular or plural, then the negation of singular is plural and the negation of plural singular. But suppose the admissible value set of the feature “gender” has more than two values: masculine, feminine and neuter. Then, the negation of the value “masculine” is one of the other two values in the complement set {feminine, neuter}, namely either feminine or neuter.

The value of a feature can, however, be complex. It can be a feature structure. In this case, the negation of a feature structure, say *F*, which is the value of some feature, is any of the feature structures that are incompatible with that feature structure *F*. Given a feature structure that has a single feature specification FEATURE0: value0, then its negation is a set of feature structures, each of which contains at least one feature specification, such that FEATURE0: ¬value0.

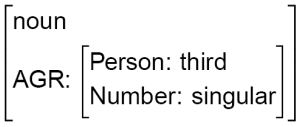
Here is an illustration:

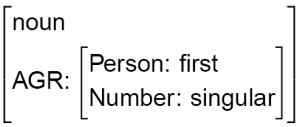
(51) Negation of a complex feature value



This feature structure carries the information about some noun that has an agreement feature specification other than that of a third person singular noun. Such a noun in English, for instance, does not have to undergo the Subject-Verb agreement as in “Mia snores” opposed to the ill-formed sentence “Mia snore”. Consider the following:

(52) Incompatible feature structures

a. 

b. 

These two are incompatible and (a) can be considered to be a particular instance of the negation of the AGR value of (b).

Negation may be represented in XML notation by means of the <**vNot**> element, as in the following example:

(53) Representing negation in XML

```
<fs type="noun">
 <f name="AGR">
  <vNot>
   <fs>
    <f name="Person">
    <symbol value="third"></symbol></f>
    <f name="Number">
    <symbol value="singular"></symbol></f>
   </fs>
  </vNot>
 </f>
</fs>
```


© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This representation is equivalent to the AVM representation (51) given above.

4.11 Informal semantics of feature structures

Semantically, a feature structure is a partial characterization of an object. It is a specification of a number of properties, where a property is described by specifying the value of a feature. For instance, an object is (partly) characterized as a plural noun by specifying that the feature POS has the value "noun" and the value NUMBER has the value "plural".

A common way to express this mathematically is by using the so-called lambda operator, which is a device for defining functions. For example, the meaning of the feature structure

(54)
$$\begin{bmatrix} \text{POS} & \text{noun} \\ \text{NUMBER} & \text{plural} \end{bmatrix}$$

can be expressed using the lambda operator as:

(55) $\lambda x : \text{pos}(x) = \text{noun} \wedge \text{number}(x) = \text{plural}$

The part of the expression that follows the initial "λ*x* :" part is called the body of the expression. Formally, such an expression defines a function that is applicable to the kind of object that *x* stands for, and that delivers the value that is obtained by substituting a specific argument for the variable *x* everywhere in the body and evaluating the body.

For instance, applied to the argument house, the expression $\lambda x : \text{pos}(x)=\text{noun} \wedge \text{number}(x)=\textit{plural}$ delivers the value "false", since pos(house)=noun but number(house)=singular, hence the conjunction is false.

A function which has true and false as its values, like the function defined by (55), is, in fact, a one-place *predicate*: if its application to an object yields true, the object in question has the property expressed by the predicate; if it yields false, it does not. The lambda expression (55) represents a complex predicate formed by conjoining the predicate of having part-of-speech noun with the predicate of having number plural. Lambda expressions are commonly used in mathematics, in formal logic, and in natural language semantics, and have a well-established model-theoretic interpretation; hence a translation of feature structures into lambda expressions can be used as a semantics of feature structures. For a formal account of the semantics of feature structures, see Annex D.

Generally speaking, the meaning of a single feature specification [*A*: *v*] is the simple predicate (or "property") of having the value *v* for the feature *A*, and the meaning of a (more complex) feature structure is the complex predicate formed by conjoining the simple predicates corresponding to the feature specifications in the FS, taking into account feature nestings and the negations, alternations, and collections of values that may occur in the feature values. Example (56), involving alternation, negation and a set-valued feature, illustrates this. This feature structure partly describes a phrase of which the head daughter is a noun of which the number feature is either singular or mass, which is not inanimate, and which takes prepositional objects using the preposition "of" or "by". This is formally expressed by the lambda expression (57). Note that, since alternation in a feature value specification expresses a choice between mutually exclusive alternatives, the corresponding predicate contains an "exclusive or", denoted by the symbol $\veebar$. An "exclusive or" construction, like $(p \veebar q)$, can be seen as an abbreviation of $((p \vee q) \wedge \neg(p \wedge q))$.

(56)
$$\begin{bmatrix} \text{POS noun} \\ \text{HDTR} \begin{bmatrix} \text{POS noun} \\ \text{NUMBER (singular}|\text{mass)} \\ \text{GENDER } \neg\text{(inanimate)} \\ \text{PREPOBS } \{\textit{of, by}\} \end{bmatrix} \end{bmatrix}$$

where HDTR stands for head daughter and PREPOBS for prepositional objects.

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(57) λx : pos(x)=noun $\wedge$ pos(hdtr(x))=noun $\wedge$ (number(hdtr(x))=singular $\veebar$ number(hdtr(x))=mass) $\wedge$

$\neg$ (gender(hdtr(x))=inanimate) $\wedge$ prepobs(hdtr(x))={“of ”, “by”}

A complication in describing the meaning of feature structures is formed by the phenomenon of re-entrancy. Re-entrancy can take three different forms, differing in the amount of information in the values of the features that share their values. Let a FS contain two features *A* and *B* that share their values. (The case of more than two features sharing their values is an easy generalization of this.) The following cases may arise:

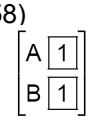
a) neither *A* nor *B* has a value specified;

b) *A* has no value specified; *B* has a value specified;

c) *A* and *B* both have a value specified.

The first case is illustrated in its simplest possible form by (58):

(58)



This says that the features *A* and *B* have the same value, whatever that value may be (or may become later in the course of a process in which the feature structure is involved). Hence it corresponds to the predicate (59):

(59) λx : $A(x) = B(x)$

The second case corresponds in its simplest possible form to a feature structure like (60):

(60)

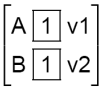
Again, the structure sharing means that the features involved should have the same value, so the predicate corresponding to the FS again contains the stipulation (59). Moreover, the FS also says that *B* has the value *v*; hence, altogether, this FS corresponds to the predicate (61):

(61) λx : $A(x) = B(x) \wedge B(x) = v$

Note that, since it is a logical consequence of $A(x)=B(x)$and $B(x)=v$ that $A(x)=v$, the latter condition does not need to be included in (61).

The case of one of the shared feature values being specified and the other unspecified, as illustrated by (60), is, in fact, a special case of the more general third case, which takes the simplest possible form (62):

(62)



If *v*1 and *v*2 in this FS are different atomic values, then the FS expresses an impossible situation and is meaningless. But if *v*1 and *v*2 are complex values, then something meaningful may be expressed if these values are compatible. This is in particular the case if *v*1 and *v*2 are feature structures that can be unified. Other possible cases arise through the use of negation or alternation, since the use of these operators corresponds to not really specifying a value of a feature, but specifying constraints on the values of the feature.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For instance, suppose *v*1 =*a* and *v*2 =*a*|*b*, where *a* and *b* are atomic values; in that case, the shared value is, in fact, *a*. Similarly, if *v*1 =*a* and *v*2 =¬*b,* the shared value is *a.*

Structure sharing can of course be combined with the use of complex values. This is illustrated in Example (63), where sharing occurs with a complex value that includes an alternation.

(63)
$$\begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{POS det} \\ \text{AGR } \boxed{1} \begin{bmatrix} \text{GENDER (feminine}|\text{masculine)} \end{bmatrix} \end{bmatrix} \\ \text{HEAD} & \begin{bmatrix} \text{POS noun} \\ \text{AGR } \boxed{1} \begin{bmatrix} \text{NUMBER plural} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Here SPEC stands for specifier and det stands for determiner. Semantically, this feature structure characterizes those phrases consisting of a determiner which has feminine or masculine gender and a plural noun, as expressed by the predicate (64):

(64) λ*x* : *pos*(*spec*(*x*)) = *det*

□ *agr*(*spec*(*x*)) = *agr*(*head*(*x*))

□ (*gender*(*agr*(*spec*(*x*))) = *feminine* □

(*gender*(*agr*(*spec*(*x*))) = *masculine*)

□ *pos*(*head*(*x*)) = *noun*

□ *number*(*agr*(*spec*(*x*))) = *plural*

5 XML representation of feature structures

5.1 Overview

This clause describes a standard for the representation of feature structures using XML, the eXtensible Markup Language, which has been officially recommended by the World Wide Web Consortium W3C as a document-processing standard for interchanging data especially over the Internet. XML is also an application of ISO 8879. It provides a rich, well-defined and platform-independent markup language for all varieties of electronic document.

5.2 Organization

The remainder of this clause is organized as follows.

— Subclause 5.3 introduces the elements <fs> and <f>, used to represent feature structures and features respectively, together with the elementary binary feature value.

— Subclause 5.4 introduces elements for representing other kinds of atomic feature values such as symbolic, numeric, and string values.

— Subclause 5.5 introduces the notion of predefined libraries or groups of features or feature values along with methods for referencing their components.

— Subclause 5.6 introduces complex values, in particular feature structures as values, thus enabling feature structures to be recursively defined.

— Subclause 5.7 treats structure sharing in feature structures.

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Subclause 5.8 discusses other complex values, in particular values which are collections, organized as sets, bags, and lists.

— Subclause 5.9 discusses how the operations of alternation, negation, and collection of feature values may be represented.

— Subclause 5.10 discusses ways of representing underspecified, default, or uncertain values. Subclause 5.11 discusses how analyses may be linked to other parts of an encoded text.

Formal definitions for all the elements introduced in this clause are provided in Annex A.

5.3 Elementary feature structures and the binary feature value

The fundamental elements used to represent a feature structure analysis are

— <f> (for feature), which represents a feature-value pair, and

— <fs> (for feature structure), which represents a structure made up of such feature-value pairs.

The <fs> element has an optional type attribute which may be used to represent typed feature structures, and may contain any number of <f> elements. An <f> element has a required name attribute and an associated value.

The value may be

— simple, that is, a single binary, numeric, symbolic (i.e. taken from a restricted set of legal values) or string value, or a collection of such values, organized in various ways, for example, as a list, or

— complex, that is, it may itself be a feature structure, thus providing a degree of recursion.

Values may be underspecified or defaulted in various ways. These possibilities are all described in more detail in this and the following subclauses.

Feature and feature-value representations (including feature structure representations) may be embedded directly at any point in an XML document, or they may be collected together in special purpose feature or feature-value libraries. The components of such libraries may then be referenced from other feature or feature-value representations, using the **feats** or **fVal** attribute as appropriate.

We begin by considering the simple case of a feature structure which contains binary-valued features only. The following three XML elements are needed to represent such a feature structure:

(65)

— **<fs>** represents a feature structure, that is, a collection of feature-value pairs organized as a structural unit. Specific attributes include:

  — **type** specifies the type of the feature structure;

  — **feats** references the feature-value specifications making up this feature structure.

— **<f>** represents a feature value specification, that is, the association of a name with a value of any of several different types. Specific attributes include:

  — **name** provides a name for the feature.

  — **fVal** references any element which can be used to represent the value of a feature.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— **<binary>** represents the value part of a feature-value specification which can contain either of exactly two possible values.

— **value** supplies a binary value (true or false, plus or minus).

The attributes **feats** and **fVal** are not discussed in this subclause: they provide an alternative way of indicating the content of an element, as further discussed in 5.6 *Feature and feature-value libraries*.

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].

NOTE Adapted from Noam Chomsky and Morris Halle (1968, p. 415)[9].

(66)
$$\begin{bmatrix} \text{consonantal} + \\ \text{vocalic} - \\ \text{voiced} - \\ \text{anterior} + \\ \text{coronal} + \\ \text{continuant} + \\ \text{strident} + \end{bmatrix}$$

This representation may be encoded in XML as follows:

(67)

```
<fs type="phonological segments">
 <f name="consonantal">
  <binary value="true"/>
 </f>
 <f name="vocalic">
  <binary value="false"/>
 </f>
 <f name="voiced">
  <binary value="false"/>
 </f>
 <f name="anterior">
  <binary value="true"/>
 </f>
 <f name="coronal">
  <binary value="true"/>
 </f>
 <f name="continuant">
  <binary value="true"/>
 </f>
 <f name="strident">
  <binary value="true"/>
 </f>
</fs>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note that <fs> elements may have an optional type attribute to indicate the kind of feature structure in question, whereas <f> elements need to have a name attribute to indicate the name of the feature. Feature structures need not be typed, but features need to be named. Similarly, the <fs> element may be empty, but the <f> element need to have (or reference) some content.

The restriction of specific features to specific types of values (e.g. the restriction of the feature “strident” to a binary value) requires additional validation, as does any restriction on the features available within a feature structure of a particular type (e.g. whether a feature structure of type phonological segment necessarily contains a feature voiced. Such validation may be carried out at the document level, using special purpose processing, at the schema level using additional validation rules, or at the declarative level, using an additional mechanism such as the feature system declaration to be treated in ISO 24610-2.

Although the term “binary” is used for this kind of value, and its representation in XML uses values such as true and false (or, equivalently, 1 and 0), it should be noted that such values are not restricted to propositional assertions. As this example shows, this kind of value is intended for use with any binary-valued feature.

Formal declarations for the <fs>, <f> and <binary> elements are provided below in Annex A.

5.4 Other atomic feature values

Features may take other kinds of atomic value. In this subclause, more elements are defined which may be used to represent: symbolic values, numeric values, and string values. The module defined by this subclause allows for the specification of additional data types if necessary, by extending the underlying class *class.single.Value*. If this is done, it is recommended that only the basic W3C datatypes should be used; more complex datatyping should be represented as feature structures.

(68)

— **<symbol>** represents the value part of a feature-value specification which contains one of a finite list of symbols. Specific attributes include:

  — **value** supplies the symbolic value for the feature, one of a finite list that may be specified in a feature declaration.

— **<numeric>** represents the value part of a feature-value specification which contains a numeric value or range. Specific attributes include:

  — **value** supplies a lower bound for the numeric value represented, and also (if max is not supplied) its upper bound;

  — **max** supplies an upper bound for the numeric value represented;

  — **trunc** specifies whether the value represented should be truncated to give an integer value.

— **<string>** represents the value part of a feature-value specification which contains a string.

The <symbol> element is 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. For example, the following might be used to represent the claim that the Latin noun form “mensas”(tables) has accusative case, feminine gender and plural number:

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(69)

```
<fs>
 <f name="case">
  <symbol value="accusative"/>
 </f>
 <f name="gender">
  <symbol value="feminine"/>
 </f>
 <f name="number">
  <symbol value="plural"/>
 </f>
</fs>
```

More formally, this representation shows a structure in which three features (case, gender and number) are used to define morpho-syntactic properties of a word. Each of these features can take one of a small number of values (for example, case can be nominative, genitive, dative, accusative, etc.) and it is therefore appropriate to represent the values taken in this instance as <symbol> elements. Note that, instead of using a symbolic value for grammatical number, one could have named the feature “singular” or “plural” and given it an appropriate binary value, as in the following example:

(70)

```
<fs>
 <f name="case">
  <symbol value="accusative"/>
 </f>
 <f name="gender">
  <symbol value="feminine"/>
 </f>
   <f name="singular">
  <binary value="false"/>
 </f>
</fs>
```

Whether one uses a binary or symbolic value in situations like this is largely a matter of taste.

The <string> element is used for the value of a feature when that value is a string drawn from a very large or potentially unbounded set of possible strings of characters, so that it would be impractical or impossible to use the <symbol> element. The string value is expressed as the content of the <string> element, rather than as an attribute value. For example, one might encode a street address as follows:

(71)

```
<fs>
 <f name="address">
  <string>3418 East Third Street</string>
 </f>
</fs>
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The <numeric> element is used when the value of a feature is a numeric value, or a range of such values. For example, one might wish to regard the house number and the street name as different features, using an encoding like the following:

(72)

```
<fs>
 <f name="houseNumber">
  <numeric value="3418"/>
 </f>
 <f name="streetName">
  <string>East Third Street</string>
 </f>
</fs>
```

If the numeric value to be represented falls within a specific range (for example an address that spans several numbers), the **max** attribute may be used to supply an upper limit:

(73)

```
<fs>
 <f name="houseNumber">
  <numeric value="3418" max="3440"/>
 </f>
 <f name="streetName">
  <string>East Third Street</string>
 </f>
</fs>
```

It is also possible to specify that the numeric value (or values) represented should (or should not) be truncated. For example, assuming that the daily rainfall in millimetres is a feature of interest for some address, one might represent this by an encoding like the following:

(74)

```
<fs>
 <f name="dailyRainFall">
  <numeric value="0.0" max="1.3" trunc="false"/>
 </f>
</fs>
```

This represents any of the infinite number of numeric values falling between 0 and 1.3; by contrast

(75)

```
<fs>
 <f name="dailyRainFall">
  <numeric value="0.0" max="1.3" trunc="true"/>
 </f>
</fs>
```

represents only two possible values: 0 and 1.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

As noted above, additional processing is necessary to ensure that appropriate values are supplied for particular features, for example to ensure that the feature “singular” is not given a value such as <symbol value=“feminine”/>. There are two ways of attempting to ensure that only certain combinations of feature names and values are used.

— First, if the total number of legal combinations is relatively small, one can predefine all of them in a construct known as a feature library, and then reference the combination required using the **feats** attribute in the enclosing <fs> element, rather than give it explicitly. 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.

— Similarly, to ensure that only feature structures containing valid combinations of feature values are used, one can put definitions for all valid feature structures inside a feature value library (so called, since a feature structure may be the value of a feature). A total of 30 feature structures ($5 \times 3 \times 2$) is required to enumerate all the possible combinations of individual case, gender and number values in the preceding illustration.

The use of such libraries and their representation in XML are further discussed in 5.5 below. However, the most general method of attempting to ensure that only legal combinations of feature names and values are used is to provide a feature system declaration to be discussed in ISO 24610-2.

5.5 Feature and feature-value libraries

As the examples in the preceding subclause suggest, the direct encoding of feature structures can be verbose. Moreover, it is often the case that particular feature-value combinations, or feature structures composed of them, are reused in different analyses. 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 feature-value specifications for 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 the same way, feature values of whatever type can be collected together in <fvLib> elements (feature-value libraries). If a feature has as its value a feature structure or other value which is predefined in this way, the **fVal** attribute may be used to point to it, as discussed in the next subclause. The following elements are used for representing feature, and feature-value libraries:

(76)

— **<fLib>** assembles a library of feature elements;

— **<fvLib>** assembles a library of reusable feature value elements (including complete feature structures).

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

(77)

```
<fLib n="phonological features">
 <f xml:id="CNS1" name="consonantal">
  <binary value="true"/>
 </f>
 <f xml:id="CNS0" name="consonantal">
  <binary value="false"/>
 </f>
 <f xml:id="VOC1" name="vocalic">
  <binary value="true"/>
 </f>
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<f xml:id="VOC0" name="vocalic">
 <binary value="false"/>
</f>
 <f xml:id="VOI1" name="voiced">
 <binary value="true"/>
</f>
<f xml:id="VOI0" name="voiced">
 <binary value="false"/>
</f>
<f xml:id="ANT1" name="anterior">
 <binary value="true"/>
</f>
<f xml:id="ANT0" name="anterior">
 <binary value="false"/>
</f>
<f xml:id="COR1" name="coronal">
 <binary value="true"/>
</f>
 <f xml:id="COR0" name="coronal">
 <binary value="false"/>
</f>
<f xml:id="CNT1" name="continuant">
 <binary value="true"/>
</f>
<f xml:id="CNT0" name="continuant">
 <binary value="false"/>
</f>
<f xml:id="STR1" name="strident">
 <binary value="true"/>
</f>
<f xml:id="STR0" name="strident">
 <binary value="false"/>
</f>
<!-- ... -->
</fLib>
```

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

(78)

```
<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"/>
```


© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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, each one represented as an identifiable <fs> element within a feature-value library, as in the following example:

(79)

```
<fvLib xml:id="fsl1" n="phonological segment definitions">
 <!-- ... -->
 <fs xml:id="T.DF" feats="#CNS1 #VOC0 #VOI0 #ANT1 #COR1 #CNT0 #STR0"/>
 <fs xml:id="D.DF" feats="#CNS1 #VOC0 #VOI1 #ANT1 #COR1 #CNT0 #STR0"/>
 <fs xml:id="S.DF" feats="#CNS1 #VOC0 #VOI0 #ANT1 #COR1 #CNT1 #STR1"/>
 <fs xml:id="Z.DF" feats="#CNS1 #VOC0 #VOI1 #ANT1 #COR1 #CNT1 #STR1"/>
 <!-- ... -->
</fvLib>
```

Once defined, these feature structure values can also be reused. Other <f> elements may invoke them by reference, using the **fVal** attribute. For example, one might use them in a feature-value pair such as:

(80)

```
<f name="dental-fricative" fVal="#T.DF"/>
```

rather than expanding the hierarchy of the component phonological features explicitly.

Feature structures stored in this way may also be associated with the text which they are intended to annotate, either by a link from the text or by means of standoff annotation techniques (see 5.11).

Note that when features or feature structures are linked to in this way, the result is effectively a copy of the item linked to into the place from which it is linked. This form of linking should be distinguished from the phenomenon of structure sharing, where it is desired to indicate that some part of an annotation structure appears simultaneously in two or more places within the structure. This kind of annotation should be represented using the <vLabel> element, as discussed in 5.7.

5.6 Feature structures as complex feature values

Features may have complex values as well as atomic ones; the simplest such complex value is represented by supplying an <fs> element as the content of an <f> element, or (equivalently) by supplying the identifier of an <fs> element as the value for the **fVal** attribute on the <f> element. Structures may be nested as deeply as appropriate, using this mechanism. 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 complex values, consider the following simple model of a word, as a structure combining surface form information, a syntactic category, and semantic information. Each word analysis is represented as an <fs type="word"> element, containing three features named surface, syntax, and semantics. The first of these has an atomic string value, but the other two have complex values, represented as nested feature structures of types "category" and "act" respectively:

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(81)

```
<fs type="word">
 <f name="surface">
  <string>love</string>
 </f>
 <f name="syntax">
  <fs type="category">
   <f name="pos">
    <symbol value="verb"/>
   </f>
   <f name="val">
    <symbol value="transitive"/>
   </f>
  </fs>
 </f>
 <f name="semantics">
  <fs type="act">
   <f name="relation">
    <symbol value="LOVE"/>
   </f>
  </fs>
 </f>
</fs>
```

This analysis does not tell us much about the meaning of the symbols “verb” or “transitive”. It might be preferable to replace these atomic feature values by feature structures. Suppose therefore that a feature-value library is maintained for each of the major syntactic categories (N, V, ADJ, PREP):

(82)

```
<fvLib n="Major category definitions">
 <!-- ... -->
 <fs xml:id="N" type="noun">
  <!--  noun features defined here -->
 </fs>
 <fs xml:id="V" type="verb">
  <!-- verb features defined here -->
 </fs>
</fvLib>
```

This library allows us to use shortcut codes (N, V, etc.) to reference a complete definition for the corresponding feature structure. Each definition may be explicitly contained within the <fs> element, as a number of <f> elements. Alternatively, the relevant features may be referenced by their identifiers, supplied as the value of the **feats** attribute, as in these examples:

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(83)

```
<!-- ... -->
 <fs xml:id="ADJ" type="adjective" feats="#N1 #V1"/>
 <fs xml:id="PREP" type="preposition" feats="#N0 #V0"/>
<!-- ... -->
```

This ability to reuse feature definitions within multiple feature structure definitions is an essential simplification in any realistic example. In this case, the existence of a feature library containing specifications for the basic feature categories is assumed like the following:

(84)

```
<fLib n="categorial features">
 <f xml:id="N1" name="nominal">
  <binary value="true"/>
 </f>
 <f xml:id="N0" name="nominal">
  <binary value="false"/>
 </f>
 <f xml:id="V1" name="verbal">
  <binary value="true"/>
 </f>
 <f xml:id="V0" name="verbal">
  <binary value="false"/>
 </f><!-- ... --></fLib>
```

With these libraries in place, and assuming the availability of similarly predefined feature structures for transitivity and semantics, the preceding example could be considerably simplified:

(85)

```
<fs type="word">
    <f name="surf">
      <string>love</string>
    </f>
    < f name="syntax">
        <fs type="category">
            <f name="pos" fVal="#V"/>
            <f name="val" fVal="#TRNS"/>
      </fs>
    </f>
    <f name="semantics">
      <fs type="act">
          <f name="rel" fVal="#LOVE"/>
      </fs>
    </f>
</fs>
```

Although in principle the **fVal** attribute could point to any kind of feature value, its use is not recommended for simple atomic values.

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.7 Re-entrant feature structures

Sometimes the same feature value is required at multiple places within a feature structure, in particular where the value is only partially specified at one or more places. The <vLabel> element is provided as a means of labelling each such reentrancy point:

(86)

— **<vLabel>** represents the value part of a feature-value specification which appears at more than one point in a feature structure;

name supplies a name for the sharing point.

For example, suppose one wishes to represent noun-verb agreement as a single feature structure. Within the representation, the feature indicating (say) number appears more than once. To represent the fact that each occurrence is another appearance of the same feature (rather than a copy), one could use an encoding like the following:

(87)

```
<fs xml:id="NVA">
 <f name="nominal">
  <fs>
   <f name="nm-num">
    <vLabel name="L1">
     <symbol value="singular"/>
    </vLabel>
   </f><!-- other nominal features -->
  </fs>
 </f>
 <f name="verbal">
  <fs>
   <f name="vb-num">
    <vLabel name="L1"/>
   </f>
  </fs><!-- other verbal features -->
 </f>
</fs>
```

In the above encoding, the features named vb-num and nm-num exhibit structure sharing. Their values, given as <vLabel> elements, are understood to be references to the same point in the feature structure, which is labelled by their name attribute.

The scope of the names used to label re-entrancy points is that of the outermost <fs> element in which they appear. When a feature structure is imported from a feature value library, or referenced from elsewhere (for example by using the **fVal** attribute), the names of any sharing points it may contain are implicitly prefixed by the identifier used for the imported feature structure, to avoid name clashes. Thus, if some other feature structure were to reference the <fs> element given in the example above, in this way:

(88)

```
<f name="class" fVal="#NVA"/>
```

then the labelled points in the example would be interpreted as if they had the name NVAL1.

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.8 Collections as complex feature values

Complex feature values need not always be represented as feature structures. Multiple values may also be organized as sets, bags (or multisets), or lists of atomic values of any type. The <vColl> element is provided to represent such cases:

(89)

— **<vColl>** represents the value part of a feature-value specification which contains multiple values organized as a set, bag, or list.

org indicates organization of given value or values as set, bag or list. Legal values are:

— **set** indicates that the given values are organized as a set;

— **bag** indicates that the given values are organized as a bag (multiset);

— **list** indicates that the given values are organized as a list.

A feature whose value is regarded as a set, bag or list may have any positive number of values as its content, or none at all (thus allowing for representation of the empty set, bag or list). The items in a list are ordered, and need not be distinct. The items in a set are not ordered, and need to be distinct. The items in a bag are neither ordered nor distinct. 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.

If no value is specified for the **org** attribute, the assumption is that the <vColl> defines a list of values. If the <vColl> element is empty, the assumption is that it represents the null list, set, or bag.

To illustrate the use of the **org** attribute, suppose that a feature structure analysis is used to represent a genealogical tree, with the information about each individual treated as a single feature structure, like this:

(90)

```
<fs xml:id="p027" type="person">
 <f name="forenames">
  <vColl>
   <string>Daniel</string>
   <string>Edouard</string>
  </vColl>
 </f>
 <f name="mother" fVal="#p002"/>
 <f name="father" fVal="#p009"/>
 <f name="birthDate">
  <fs type="date" feats="#y1988 #m04 #d17"/>
 </f>
 <f name="birthPlace" fVal="#austintx"/>
 <f name="siblings">
  <vColl org="set">
   <fs copyOf="#pnb005"/>
   <fs copyOf="$prb001"/>
  </vColl>
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

In this example, the <vColl> element is first used to supply a list of name feature values, which together constitute the forenames feature. Other features are defined by reference to values which are assumed to be held in some external feature value library (not shown here). For example, the <vColl> element is used a second time to indicate that the person's siblings should be regarded as constituting a set rather than a list. Each sibling is represented by a feature structure. In this example, each feature structure is a copy of one specified in the feature value library.

If a specific feature contains only a single feature structure as its value, the component features of which are organized as a set, bag or list, it may be more convenient to represent the value as a <vColl> rather than as a <fs>. 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.

(91)

```
<fs type="word">
 <f name="category">
  <symbol value="verb"/>
 </f>
 <f name="tense">
  <symbol value="present"/>
 </f>
 <f name="agreement">
  <fs>
   <f name="person">
    <symbol value="third"/>
   </f>
   <f name="number">
    <symbol value="singular"/>
   </f>
  </fs>
 </f>
</fs>
```

If the names of the features contained within the agreement feature structure are of no particular significance, the following simpler representation may be used:

(92)

```
<fs type="word">
 <f name="category">
  <symbol value="verb"/>
 </f>
 <f name="tense">
  <symbol value="present"/>
 </f>
 <f name="agreement">
  <vColl org="set">
```


© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      <symbol value="third"/>
      <symbol value="singular"/>
     </vColl>
    </f>
   </fs>
```

The <vColl> element is also useful in cases where an analysis has several components. In the following example, the French word “auxquels” has a two-part analysis, represented as a list of two values. The first specifies that the word contains a preposition; the second, that it contains a masculine plural relative pronoun:

(93)

```
<fs>
 <f name="word">
  <symbol value="auxquels"/>
 </f>
 <f name="morpho-syntatic feature">
  <vColl org="list">
   <fs>
    <f name="category">
     <symbol value="preposition"/>
    </f>
   </fs>
   <fs>
    <f name="category">
     <symbol value="pronoun"/>
    </f>
    <f name="kind">
     <symbol value="relative"/>
    </f>
    <f name="number">
     <symbol value="plural"/>
    </f>
    <f name="gender">
     <symbol value="masculine"/>
    </f>
   </fs>
  </vColl>
 </f>
</fs>
```

The set, bag or list which has no members is known as the null (or empty) set, bag or list. A <vColl> element with no content and with no value for its **feats** attribute is interpreted as referring to the null set, bag, or list, depending on the value of its **org** attribute.

If, for example, the individual described by the feature structure with identifier #p027 in (90) had no siblings, the “siblings” feature might be specified as follows.

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(94)

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

A <vColl> element may also collect together one or more other <vColl> elements, if, for example, one of the members of a set is itself a set, or if two lists are concatenated together. Note that such collections pay no attention to the contents of the nested <vColl> elements: if it is desired to produce the union of two sets, the <vMerge> element discussed below should be used to make a new collection from the two sets.

5.9 Feature value expressions

5.9.1 Overview

It is sometimes desirable to express the value of a feature as the result of an operation over some other value (for example, as “not green”, or as “male or female”, or as the concatenation of two collections). Three special purpose elements are provided to represent disjunctive alternation, negation, and collection of values:

(95)

— **<vAlt>** represents the value part of a feature-value specification which contains a set of values, only one of which can be valid;

— **<vNot>** represents a feature value which is the negation of its content;

— **<vMerge>** represents a feature value which is the result of merging together the feature values contained by its children, using the organization specified by the **org** attribute, which indicates the organization of the resulting merged values as set, bag or list.

5.9.2 Alternation

The <vAlt> element can be used wherever a feature value can appear. It contains two or more feature values, any one of which is to be understood as the value required. Suppose, for example, that you are using a feature system to describe residential property, using such features as “number.of.bathrooms”. In a particular case, you might wish to represent uncertainty as to whether a house has two or three bathrooms. As shown above, one simple way to represent this would be with a numeric maximum:

(96)

```
<f name="number.of.bathrooms">
  <numeric value="2" max="3"/>
</f>
```

A better, and more general, way would be to represent the alternation explicitly, in this way:

(97)

```
<f name="number.of.bathrooms">
  <vAlt>
   <numeric value="2"/>
   <numeric value="3"/>
  </vAlt>
</f>
```

The <vAlt> element represents alternation over feature values, not feature-value pairs. If, therefore, the uncertainty relates to two or more feature-value specifications, each needs to be represented as a feature

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO 2006 – All rights reserved
Not for Resale

structure, since a feature structure can always appear where a value is required. For example, suppose that it is uncertain as to whether the house being described has two bathrooms or two bedrooms, a structure like the following may be used:

(98)

```
<f name="rooms">
 <vAlt>
  <fs>
   <f name="number.of.bathrooms">
    <numeric value="2"/>
   </f>
  </fs>
  <fs>
   <f name="number.of.bedrooms">
    <numeric value="2"/>
   </f>
  </fs>
 </vAlt>
</f>
```

Note that alternation is always regarded as exclusive: in the case above, the implication is that having two bathrooms excludes the possibility of having two bedrooms and vice versa. If inclusive alternation is required, a <vColl> element may be included in the alternation as follows:

(99)

```
<f name="rooms">
 <vAlt>
  <fs>
   <f name="number.of.bathrooms">
    <numeric value="2"/>
   </f>
  </fs>
  <fs>
   <f name="number.of.bedrooms">
    <numeric value="2"/>
   </f>
  </fs>
  <vColl>
   <fs>
    <f name="number.of.bathrooms">
     <numeric value="2"/>
    </f>
   </fs>
   <fs>
    <f name="number.of.bedrooms">
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        <numeric value="2"/>
      </f>
     </fs>
    </vColl>
   </vAlt>
  </f>
```

This analysis indicates that the property may have two bathrooms, two bedrooms, or both two bathrooms and two bedrooms.

As the previous example shows, the <vAlt> element can also be used to indicate alternations among values of features organized as sets, bags or lists. Suppose you use a feature “selling.points” to describe 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 selling points include an alarm system and a good view, and either a pool or a jacuzzi (but not both). This situation could be represented, using the <vAlt> element, as follows.

(100)

```
<fs type="real estate listing">
 <f name="selling.points">
  <vColl org="set">
   <string>alarm system</string>
   <string>good view</string>
   <vAlt>
    <string>pool</string>
    <string>jacuzzi</string>
   </vAlt>
  </vColl>
 </f>
</fs>
```

Now suppose the situation is like the preceding except that one is also uncertain whether the property has an alarm system or a good view. This can be represented as follows.

(101)

```
<fs type="real estate listing">
 <f name="selling.points">
  <vColl org="set">
   <vAlt>
    <string>alarm system</string>
    <string>good view</string>
   </vAlt>
   <vAlt>
    <string>pool</string>
    <string>jacuzzi</string>
   </vAlt>
     </vColl>
 </f>
</fs>
```


© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If a large number of ambiguities or uncertainties need to be represented, involving a relatively small number of features and values, it is recommended that a standoff technique, for example using the general purpose <alt> element discussed in the *TEI Guidelines P5* [50], be used, rather than the special purpose <vAlt> element.

5.9.3 Negation

The <vNot> element can be used wherever a feature value can appear. It contains any feature value and returns the complement of its contents. For example, the feature “number.of.bathrooms" in the following example has any whole numeric value other than 2:

(102)

```
<f name="number.of.bathrooms">
 <vNot>
  <numeric value="2"/>
 </vNot>
</f>
```

Strictly speaking, the effect of the <vNot> element is to provide the complement of the feature values it contains, rather than their negation. If a feature system declaration is available which defines the possible values for the associated feature, then it is possible to say more about the negated value. For example, suppose that the available values for the feature “case” are declared to be nominative, genitive, dative, or accusative, whether in a TEI feature system declaration, ISO 24610-2 or by some other means. Then the following two specifications are equivalent:

(103)

```
a) <f name="case">
  <vNot>
   <symbol value="genitive"/>
  </vNot>
</f> ;
b) <f name="case">
  <vAlt>
   <symbol value="nominative"/>
   <symbol value="dative"/>
   <symbol value="accusative"/>
  </vAlt>
</f> .
```

If, however, no such system declaration is available, all that one can say about a feature specified via negation is that its value is something other than the negated value.

Negation is always applied to a feature value, rather than to a feature-value pair. The negation of an atomic value is the set of all other values which are possible for the feature.

Any kind of value can be negated, including collections (represented by <vColl> elements) or feature structures (represented by <fs> elements). The negation of any complex value is understood to be the set of values which cannot be unified with it. Thus, for example, the negation of the feature structure *F* is understood to be the set of feature structures which are not unifiable with *F*. In the absence of a constraint mechanism such as the feature system declaration, the negation of a collection is anything that is not unifiable with it, including collections of different types and atomic values. It will generally be more useful to require that the

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

organization of the negated value be the same as that of the original value, for example that a negated set is understood to mean the set which is a complement of the set, but such a requirement cannot be enforced in the absence of a constraint mechanism.

5.9.4 Collection of values

The <vMerge> element can be used wherever a feature value can appear. It contains two or more feature values, all of which are to be collected together. The organization of the resulting collection is specified by the value of the **org** attribute, which need not necessarily be the same as that of its constituent values if these are collections. For example, one can change a list to a set, or vice versa.

As an example, suppose that one wishes to represent the range of possible values for a feature genders used to describe some language. It would be natural to represent the possible values as a set, using the <vColl> element as in the following example:

(104)

```
<fs>
 <f name="genders">
  <vColl org="set">
   <symbol value="masculine"/>
   <symbol value="feminine"/>
  </vColl>
 </f>
</fs>
```

Suppose however that it is discovered for some language that it is necessary to add a new possible value, and to treat the value of the feature as a list rather than as a set. The <vMerge> element can be used to achieve this:

(105)

```
<fs>
 <f name="genders">
  <vMerge org="list">
   <vColl org="set">
    <symbol value="masculine"/>
    <symbol value="feminine"/>
   </vColl>
   <symbol value="neuter"/>
  </vMerge>
 </f>
</fs>
```

5.10 Default values

The value of a feature may be underspecified in a number of different ways. It may be null, unknown, or uncertain with respect to a range of known possibilities, as well as being defined as a negation or an alternation. As previously noted, the specification of the range of known possibilities for a given feature is not part of the current specification. In the TEI scheme or ISO 24610-2, this information is conveyed by the feature system declaration. Using this, or some other system, might specify (for example) that the range of values for an element includes symbols for masculine, feminine, and neuter, and that the default value is neuter. With

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

such definitions available to us, it becomes possible to say that some feature takes the default value, or some unspecified value from the list. The following element is provided for this purpose:

(106)

— **<default>** represents the value part of a feature-value specification which contains a defaulted value.

The value of an empty <f> element which also lacks an **fVal** attribute is understood to be the most general case, i.e. any of the available values. Thus, assuming the feature system defined above, the following two representations a) and b) are equivalent.

(107)

a) <f name="gender"/>
b) <f name="gender">
<vAlt>
<symbol value="feminine"/>
<symbol value="masculine"/>
<symbol value="neuter"/>
</vAlt>
</f>

If, however, the value is explicitly stated to be the default one, using the <default> element, then the following two representations are equivalent:

(108)

<f name="gender">
<default/>
</f>

(109)

<f name="gender">
<symbol value="neuter"/>
</f>

Similarly, if the value is stated to be the negation of the default, then the following two representations are equivalent:

(110)

<f name="gender">
<vNot>
<default/>
</vNot>
</f>

(111)

<f name="gender">
<vAlt>
<symbol value="feminine"/>
<symbol value="masculine"/>

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  </vAlt>
</f>
```

5.11 Linking text and analysis

Text elements can be linked with feature structures using any of the linking methods discussed elsewhere in *The TEI Guidelines*. In the simplest case, the **ana** attribute may be used to point from any element to an annotation of it, as in the following example:

(112)

```
<s n="00741">
  <w ana="#at0">The</w>
  <w ana="#ajs">closest</w>
  <w ana="#pnp">he</w>
  <w ana="#vvd">came</w>
  <w ana="#prp">to</w>
  <w ana="#nn1">exercise</w>
  <w ana="#vbd">was</w>
  <w ana="#to0">to</w>
  <w ana="#vvi">open</w>
  <w ana="#crd">one</w>
  <w ana="#nn1">eye</w>
  <phr ana="#av0">
   <w>every</w>
   <w>so</w>
   <w>often</w>
  </phr>
  <c ana="#pun">,</c>
  <w ana="#cjs">if</w>
  <w ana="#pni">someone</w>
  <w ana="#vvd">entered</w>
  <w ana="#at0">the</w>
    <w ana="#nn1">room</w>
    <!-- ... -->
</s>
```

The values specified for the **ana** attribute reference components of a feature structure library, which represents all of the grammatical structures used by this encoding scheme. (For illustrative purposes, only the structures needed for the first six words are cited.)

(113)

```
<fvLib xml:id="C6" n="Clause 6 POS Codes">
<!-- ... -->
  <fs xml:id="ajs" feats="#wj #ds"/>
  <fs xml:id="at0" feats="#wl"/>
  <fs xml:id="pnp" feats="#wr #rp"/>
  <fs xml:id="vvd" feats="#wv #bv #fd"/>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
 <fs xml:id="prp" feats="#wp #bp"/>
 <fs xml:id="nn1" feats="#wn #tc #ns"/><!-- ... -->
</fsLib>
```

The components of each feature structure in the library are referenced in much the same way, using the **feats** attribute to identify one or more <f> elements in the following feature library (again, only a few of the available features are quoted here):

(114)

```
<fLib><!-- ... -->
 <f xml:id="bv" name="verb-base">
  <symbol value="main"/>
 </f>
 <f xml:id="bp" name="prep-base">
  <symbol value="lexical"/>
 </f>
 <f xml:id="ds" name="degree">
  <symbol value="superlative"/>
 </f>
 <f xml:id="fd" name="verb-form">
  <symbol value="ed"/>
 </f>
 <f xml:id="ns" name="number">
  <symbol value="singular"/>
 </f>
 <f xml:id="rp" name="pronoun-type">
  <symbol value="personal"/>
 </f>
 <f xml:id="tc" name="noun-type">
  <symbol value="common"/>
 </f>
 <f xml:id="wj" name="class">
  <symbol value="adjective"/>
 </f>
 <f xml:id="wl" name="class">
  <symbol value="article"/>
 </f>
 <f xml:id="wn" name="class">
  <symbol value="noun"/>
 </f>
 <f xml:id="wp" name="class">
  <symbol value="preposition"/>
 </f>
 <f xml:id="wr" name="class">
  <symbol value="pronoun"/>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  </f>
  <f xml:id="wv" name="class">
   <symbol value="verb"/>
  </f><!-- ... -->
 </fLib>
```

Alternatively, a standoff technique may be used, as in the following example, where a <linkGrp> element is used to link selected characters in the text “Caesar seized control” with their phonological representations.

(115)

```
<text><!-- ... -->
 <body><!-- ... -->
    <s xml:id="S1">
    <w xml:id="S1W1">
    <c xml:id="S1W1C1">C</c>ae<c xml:id="S1W1C2">s</c>ar</w>
    <w xml:id="S1W2">
    <c xml:id="S1W2C1">s</c>ei<c xml:id="S1W2C2">z</c>e<c xml:id="S1W2C3">d</c>
    </w>
     <w xml:id="S1W3">con<c xml:id="S1W3C1">t</c>rol</w>.
   </s><!-- ... -->
 </body>
 <fvLib xml:id="FSL1" n="phonological segment definitions">
<!-- as in previous example -->
 </fvLib>
   <linkGrp type="phonology">
   <!-- ... -->
    <link targets="#S.DF #S1W3C1"/>
     <link targets="#Z.DF #S1W2C3"/>
     <link targets="#S.DF #S1W2C1"/>
   <link targets="#Z.DF #S1W2C2"/>
 <!-- ... -->
  </linkGrp>
</text>
```

As this example shows, a standoff solution requires that every component to be linked to shall bear an identifier. To handle the POS tagging example above, therefore, each annotated element would need an identifier of some sort, as follows:

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(116)

```
<s xml:id="mds09" n="00741">
 <w xml:id="mds0901">The</w>
 <w xml:id="mds0902">closest</w>
 <w xml:id="mds0903">he</w>
 <w xml:id="mds0904">came</w>
 <w xml:id="mds0905">to</w>
 <w xml:id="mds0906">exercise</w><!-- ... -->
</s>
```

It would then be possible to link each word to its intended annotation in the feature library quoted above, as follows:

(117)

```
<linkGrp type="POS-codes">
<!-- ... -->
 <link targets="#mds0901 #at0"/>
 <link targets="#mds0902 #ajs"/>
 <link targets="#mds0903 #pnp"/>
 <link targets="#mds0904 #vvd"/>
 <link targets="#mds0905 #prp"/>
 <link targets="#mds0906 #nn1"/>
 <link targets="#mds0907 #vbd"/>
 <link targets="#mds0908 #to0"/>
 <link targets="#mds0909 #vvi"/>
 <link targets="#mds0910 #crd"/>
<!-- ... -->
</linkGrp>
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(informative)

Formal definitions and implementation of the XML representation of feature structures

A.1 Overview

The elements discussed in this chapter constitute a module of the TEI scheme which, like other modules, may be expressed using a variety of schema languages. This annex provides a formal definition for the whole module, using the compact syntax defined for ISO 19757-2. This is followed by formal documentation for each element defined by the module, identical to that provided by *The TEI Guidelines.*

A.2 RELAX NG specification for the module

Module : iso-fs-standalone.rnc

namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

Schema generated 2005-09-29T14:00:12+02:00

Copyright 2005 TEI Consortium. This is free software; you can redistribute it and/or modify it under the terms of # the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, # or (at your option) any later version. This material is distributed in the hope that it will be useful, but WITHOUT # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A # PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a # copy of the GNU General Public License along with this file; if not, write to the Free Software Foundation, Inc., # 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

To contact the TEI Consortium use the following addresses.

For general (non-technical) enquiries: tei@tei-c.org

For membership enquiries: membership@tei-c.org

For technical enqiries: editors@tei-c.org

For the current release of the TEI schema visit http://www.tei-c.org/release/xml/tei/schema/

1. classes

The following declarations have been copied here from tei.rnc to make this module self-contained.

Definitions from module tei

1. classes

tei.global = notAllowed

tei.linking |= tei.global

tei.analysis |= tei.global

tei.global.attributes =

tei.global.attribute.xmlid,

tei.global.attribute.n,

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  tei.global.attribute.xmllang,
  tei.global.attribute.rend,
  tei.global.attribute.xmlbase
tei.global.attribute.xmlid =
  ## provides a unique identifier for the element bearing the attribute.
  attribute xml:id { xsd:ID }?
tei.global.attribute.n =
  ## gives a number (or other label) for an element, which is not necessarily unique within the document.
  attribute n { text }?
tei.global.attribute.xmllang =
  ## indicates the language of the element content using the codes from RFC 3066
  attribute xml:lang { tei.data.language }?
tei.global.attribute.rend =
  ## indicates how the element in question was rendered or presented in the source text.
  attribute rend { text }?
tei.global.attribute.xmlbase =
  ## provides a base URI reference with which applications can resolve relative URI references into absolute URI  ##
  references.
  attribute xml:base { tei.data.pointer }?

tei.featureVal |= tei.complexVal
tei.singleVal = notAllowed
tei.featureVal |= tei.singleVal
tei.featureVal = notAllowed
tei.metadata = notAllowed

tei.data.name = xsd:Name
tei.data.language = xsd:language
tei.data.numeric = xsd:double
tei.data.truthValue = xsd:boolean
tei.data.pointer = xsd:anyURI
tei.data.pointers = list { tei.data.pointer+ }
tei.data.enumerated = xsd:token

# Definitions from module iso-fs

# 2. elements

binary =
  ## (binary value) represents the value part of a feature-value specification which can contain either of exactly   ## two
  possible values.
  element binary { binary.content, binary.attributes }
binary.content = empty
binary.attributes =
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  tei.global.attributes,
  ## supplies a binary value.
  attribute value { tei.data.truthValue },
  [ a:defaultValue = "binary" ] attribute TEIform { text }?
tei.singleVal |= binary

\default =
  ## (Default feature value) represents the value part of a feature-value specification which contains a defaulted ## value.
  element default { default.content, default.attributes }
default.content = empty
default.attributes =
  tei.global.attributes,
  [ a:defaultValue = "default" ] attribute TEIform { text }?
tei.singleVal |= \default

f =
  ## (Feature) represents a feature value specification, that is, the association of a name with a value of any of   ## several
  different types.
  element f { f.content, f.attributes }
f.content = tei.featureVal*
f.attributes =
  tei.global.attributes,
  ## provides a name for the feature.
  attribute name { tei.data.name },
  ## references any element which can be used to represent the value of a feature.
  attribute fVal { tei.data.pointer }?,
  [ a:defaultValue = "f" ] attribute TEIform { text }?

fLib =
  ## (Feature library) assembles a library of feature elements.
  element fLib { fLib.content, fLib.attributes }
fLib.content = f+
fLib.attributes =
  tei.global.attributes,
  [ a:defaultValue = "fLib" ] attribute TEIform { text }?
tei.metadata |= fLib

fs =
  ## (Feature structure) represents a feature structure, that is, a collection of feature-value pairs organized as a
  ## structural unit.
  element fs { fs.content, fs.attributes }
fs.content = f*
fs.attributes =
```


© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  tei.global.attributes,
  ## specifies the type of the feature structure.
  attribute type { tei.data.enumerated }?,
  ## references the feature-value specifications making up this feature structure.
  attribute feats { tei.data.pointers }?,
  [ a:defaultValue = "fs" ] attribute TEIform { text }?
tei.complexVal |= fs
tei.metadata |= fs

fvLib =
  ## (Feature-value library) assembles a library of reusable feature value elements (including complete feature   ##
  structures).
  element fvLib { fvLib.content, fvLib.attributes }
fvLib.content = (tei.featureVal)*
fvLib.attributes =
     tei.global.attributes,
    [ a:defaultValue = "fvLib" ] attribute TEIform { text }?
tei.metadata |= fvLib

numeric =
  ## (Numeric value) represents the value part of a feature-value specification which contains a numeric value or ## range.
  element numeric { numeric.content, numeric.attributes }
numeric.content = empty
numeric.attributes =
  tei.global.attributes,
  ## supplies a lower bound for the numeric value represented, and also (if max is not supplied) its upper bound. ##
  attribute value { tei.data.numeric }, supplies an upper bound for the numeric value represented.
  attribute max { tei.data.numeric }?,
  ## specifies whether the value represented should be truncated to give an integer value.
  attribute trunc { tei.data.truthValue }?,
  [ a:defaultValue = "numeric" ] attribute TEIform { text }?
tei.singleVal |= numeric

\string =
  ## (String value) represents the value part of a feature-value specification which contains a string.
  element string { string.content, string.attributes }
string.content = text
string.attributes =
  tei.global.attributes,
  [ a:defaultValue = "string" ] attribute TEIform { text }?
tei.singleVal |= \string

symbol =
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  ## (Symbolic value) represents the value part of a feature-value specification which contains one of a finite list of ## symbols.
  element symbol { symbol.content, symbol.attributes }
symbol.content = empty
symbol.attributes =
  tei.global.attributes,
  ## supplies the symbolic value for the feature, one of a finite list that may be specified in a feature declaration.
  attribute value { tei.data.name },
  [ a:defaultValue = "symbol" ] attribute TEIform { text }?
tei.singleVal |= symbol

vAlt =
  ## (Value alternation) represents the value part of a feature-value specification which contains a set of values, ## only one of which can be valid.
  element vAlt { vAlt.content, vAlt.attributes }
vAlt.content = tei.featureVal, tei.featureVal+
vAlt.attributes =
  tei.global.attributes,
  [ a:defaultValue = "vAlt" ] attribute TEIform { text }?
tei.singleVal |= vAlt

vColl =
  ## (collection of values) represents the value part of a feature-value specification which contains multiple values ## organized as a set, bag, or list.
  element vColl { vColl.content, vColl.attributes }
vColl.content = (fs | tei.singleVal)*
vColl.attributes =
  tei.global.attributes,
  ## indicates organization of given value or values as set, bag or list.
  attribute org {
    ## (indicates that the given values are organized as a set. )
    "set" | ## (indicates that the given values are organized as a bag (multiset). )
      "bag" | ## (indicates that the given values are organized as a list.)
      "list"
  }?,
  [ a:defaultValue = "vColl" ] attribute TEIform { text }?
tei.complexVal |= vColl

vLabel =
  ## (value label) represents the value part of a feature-value specification which appears at more than one point ## in a feature structure
  element vLabel { vLabel.content, vLabel.attributes }
vLabel.content = tei.featureVal?
vLabel.attributes =
  tei.global.attributes,
```


© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  ## supplies a name for the sharing point.
  attribute name { tei.data.name },
  [ a:defaultValue = "vLabel" ] attribute TEIform { text }?
tei.singleVal |= vLabel

vMerge =
  ## (Merged collection of values) represents a feature value which is the result of merging together the feature  ## values
  contained by its children, using the organization specified by the ORG attribute.
  element vMerge { vMerge.content, vMerge.attributes }
vMerge.content = tei.featureVal+
vMerge.attributes =
  tei.global.attributes,
  ## indicates the organization of the resulting merged values as set, bag or list.
  attribute org {
    ## (indicates that the resulting values are organized as a set.)
    "set" | ## (indicates that the resulting values are organized as a bag (multiset).)
      "bag" | ## (indicates that the resulting values are organized as a list.)
      "list"
  }?,
  [ a:defaultValue = "vMerge" ] attribute TEIform { text }?
tei.complexVal |= vMerge

vNot =
  ## (Value negation) represents a feature value which is the negation of its content.
  element vNot { vNot.content, vNot.attributes }
vNot.content = tei.featureVal
vNot.attributes =
  tei.global.attributes,
  [ a:defaultValue = "vNot" ] attribute TEIform { text }?
tei.complexVal |= vNot
```

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(informative)

Examples for illustration

Consider the problem of specifying the grammatical case, gender and number features of classical Latin noun forms. Assuming that the case "feature" can take on any of the six values "nominative", "vocative", "genitive", "dative", "accusative" and "ablative", 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 Latin word "rosas" meaning "roses" has accusative case, feminine gender and plural number.

```
<fs type="word structure">
  <f name="case">   <symbol value="accusative"/> </f>
  <f name="gender"> <symbol value="feminine"/> </f>
  <f name="number"> <symbol 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">   <symbol value="feminine"/> </f>
  <f name="gender"> <symbol value="accusative"/> </f>
  <f name="number"> <binary value="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 ($5 \times 3 \times 2$) 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 **feats** 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 Annex A; the <vAlt> element is discussed in 5.9.2.

```
<!-- VALRANGE specification for CASE feature --> <valRange>
  <vAlt>
    <symbol value="nominative"/>
       <symbol value="vocative"/>
    <symbol value="genitive"/>
    <symbol value="dative"/>
```


© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      <symbol value="accusative"/>
      <symbol value="ablative"/>
    </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. 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.

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C
(informative)

Type inheritance hierarchies

C.1 Overview

Types are used organize feature structures into natural classes. It is convenient to think of them as being organized into an inheritance hierarchy based on their generality; in this capacity, they perform much the same role as concepts in terminological knowledge representation systems or abstract datatypes in object oriented-programming languages. The type inheritance hierarchy is defined by assuming a finite set **type** of types, ordered according to their specificity, where type τ is more specific than type σ if τ inherits all properties and characteristics from σ. In this case, σ subsumes or is more general than τ: for σ, τ in **type**, σ subsumes τ. If σ subsumes τ, then σ is also said to be a supertype of τ, or, inversely, τ is a subtype of σ.

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

C.2 Definition

A type hierarchy forms a treelike finite structure. It shall have the following properties.

(118) Properties of type hierarchy

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

— **No cycle:** It shall have no cycles.

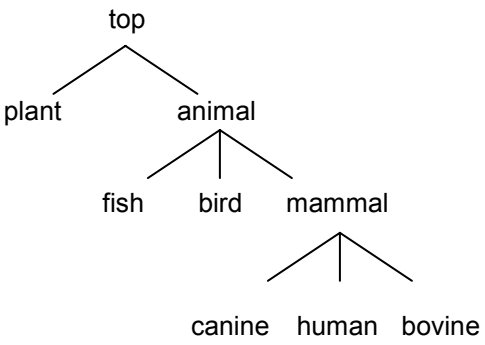
— **Unique greatest lower bounds:** Any two compatible types shall have a unique highest common descendant or subtype called greatest lower bound. Incompatible types share no common descendants or subtypes.

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

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

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(119) Type hierarchy for some animals top

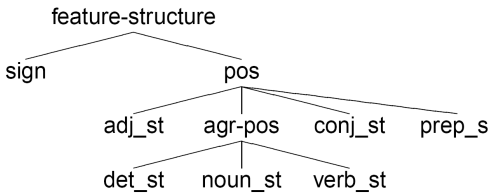


Here, while the types human and canine are not compatible, the types animal and human are compatible and thus shall have a unique greatest lower bound. Being in the hierarchical relation, the type “human” becomes that lower bound in a trivial manner.

A linguistically more relevant example can be given as below:

NOTE Taken modified from Sag, Wasow and Bender (2003, p. 61)[47].

(120) Linguistic example for type hierarchy feature structure



Here the type “feature structure” is treated as the unique top type. The types det_st, noun_st and verb_st are treated as subtypes of the type agr-pos, since they are governed by agreement rules in English.

In a language like French or Latin, the type adj_st should be also be treated as a subtype of agr-pos.

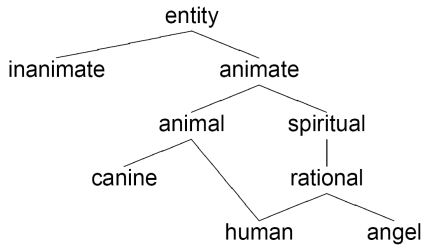
© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

C.3 Multiple inheritance

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

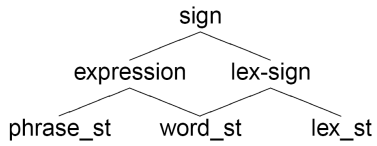
(121) Medieval hierarchy of entities



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

Linguistic signs may also allow multiple inheritance like the following.

(122) Multiple inheritance sign



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

C.4 Type constraints

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

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

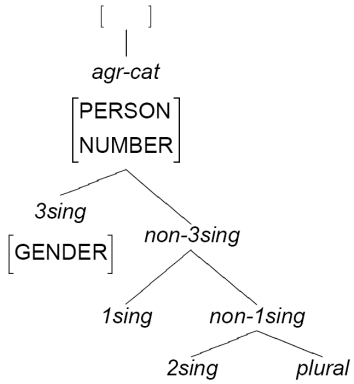
Consider features like GENDER and AUX for English. The feature GENDER is appropriate for the type noun_st, while it is not so for the type verb_st. Likewise, the feature AUX is appropriate for the type verb_st, but not for the type noun_st. Hence, each type is closely associated with a set of appropriate features.

The values of each feature are again restricted as types. The appropriate or permissible values of the feature GENDER are feminine, masculine, neuter, etc., but cannot be boolean or binary values, namely + and -. On the other hand, the appropriate values of the feature AUX are only Boolean values.

Consider the following type hierarchy for agreement [copied from the type hierarchy presented in Sag, Wasow and Bender (2003, p. 492)][47].

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(123) Agreement type hierarchy



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

Furthermore, when each feature is introduced, it is also associated with a particular set of its admissible values. The following would be an example:

(124) Admissible values

Features	**Admissible values**
PERSON	{1st, 2nd, 3rd}
NUMBER	{singular, plural}
GENDER	{feminine, masculine, neuter}

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

(125) Ill-formed feature structure

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

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

(126) Well-formed feature structure

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

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

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex D
(informative)

Denotational semantics of feature structure

D.1 Feature structure signatures

A feature structure signature is given by a triple $\Sigma = (\mathbf{Feat}, \mathbf{Type}, \mathbf{Atom}, \mathbf{Var})$ where

— **Feat** denotes a set of features

— **Type** denotes a set of types including a distinguished type $\top$

— **Atom** denotes a set of atomic values

— **Var** denotes a set of labels to be used to name sharing points

NOTE Untyped FSs correspond to having only one type, namely $\top$.

D.2 Feature structure algebra

From a syntactic point of view, the algebra *FS* of a Feature Structure over a signature Σ is defined inductively using the following constructors.

NOTE This linear notation can obviously be replaced by a more standard AVM notation.

(atomic) $\mathbf{Atom} \subset \mathcal{FS}$

(features) For all $\tau \in \mathbf{Type}$, $(f_1, \ldots, f_n) \in \mathbf{Feat}$, $(v_1, \ldots, v_n) \in \mathcal{FS}$, $\tau[f_1 : v_1 \ \ldots \ f_n : v_n] \in \mathcal{FS}$

(alternation) For all $v, w \in \mathcal{FS}$, $v|w \in \mathcal{FS}$

(negation) For all $v \in \mathcal{FS}$, $\neg v \in \mathcal{FS}$

(sharing) For all $x \in \mathbf{Var}$, $v \in \mathcal{FS}$, $\boxed{\mathrm{X}}v \in \mathcal{FS}$

(collections) For all $(v_1, \ldots, v_n) \in \mathcal{FS}^n$, $\begin{cases} \mathrm{list}\{v_1, \ldots, v_n\} \in \mathcal{FS} \\ \mathrm{set}\{v_1, \ldots, v_n\} \in \mathcal{FS} \\ \mathrm{bag}\{v_1, \ldots, v_n\} \in \mathcal{FS} \end{cases}$

(merging) For all $v, w \in \mathcal{FS}$, $v \oplus w \in \mathcal{FS}$

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.3 FS domains

Given some set $\mathcal{U}$, we consider the set $\mathcal{U}^\omega = lim_{n \rightsquigarrow \omega} I^n(\mathcal{U})$, built by limit of the following operator I:

$$I(D) = D \cup D^\star \cup \text{finite}(\mathbb{N}^D)$$

where

— $D^\star$ stands for all finite tuple $(x_1, \ldots, x_n)$ of elements of D, including the empty tuple $()$.

— $\text{finite}(\mathbb{N}^D)$ stands for all functions $\rho : D \mapsto \mathbb{N}$ with a finite domain $\{x \in D | \rho(x) > 0\}$. Such a mapping may be represented by $\{x_1 : \rho(x_1), \ldots, x_n : \rho(x_n)\}$, including the empty mapping $\{\}$.

Mapping may be combined using (with some slight abuse) the “union” operator $\cup$ defined by $(\rho \cup \rho')(x) = \rho(x) + \rho'(x)$ for all $x \in D$.

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.4 Feature structure interpretations

Feature structures provide partial information about objects in some given universe.

More formally, an Σ-interpretation I (or model) is a 4-uple $(\mathcal{U}, I_{\mathbf{Type}} : \mathbf{Type} \mapsto 2^{\mathcal{U}}, I_{\mathbf{Feat}} : \mathbf{Feat} \mapsto \mathcal{U} \mapsto \mathcal{U}^{\omega}, I_{\mathbf{Atom}} : \mathbf{Atom} \mapsto \mathcal{U})$ and the interpretation $I(s)$ of a feature structure s is defined by

$$I(s) = \bigcup_{\mu} I(s,\mu)$$

where μ denotes a valuation over variables, i.e. a mapping $\mathbf{Var} \mapsto 2^{\mathcal{U}^{\omega}}$, and $I(s,\mu)$ an interpretation wrt μ defined by

— $I(a) = \{ I_{\mathbf{Atom}}(a) \}$, for all $a \in \mathbf{Atom}$

— $I(\tau[(f_i : v_i)_{i=1...n}]) = \{x \in I_{\mathbf{Type}}(\tau) | \forall f_i,\ I(f_i)(x) \in I(v_i,\mu)\}$

— $I(v|w,\mu) = I(v,\mu) \cup I(w,\mu)$

— $I(\neg v,\mu) = \mathcal{U} - I(v,\mu)$

— $I(\boxed{\mathrm{X}}v,\mu) = \mu(x) \cap I(v,\mu)$

— $I({}_{\mathrm{list}}\{v_1,\ldots,v_n\},\mu) = (I(v_1,\mu),\ldots,I(v_n,\mu))$

— $I({}_{\mathrm{set}}\{v_1,\ldots,v_n\},\mu) = \{I(v_1,\mu) : 1,\ldots,I(v_n,\mu) : 1\}$

— $I({}_{\mathrm{bag}}\{v_1,\ldots,v_n\},\mu) = \{I(v_1,\mu) : 1\} \cup \ldots \cup \{I(v_n,\mu) : 1\}$

— $I(v \oplus w,\mu) = I(v,\mu) \oplus I(w,\mu)$

For all $(x,y) \in \mathcal{U}^2$, the operation $x \oplus y$ is defined by:

$$x \oplus y = \begin{cases} (x_1,\ldots,x_n,y_1,\ldots,y_n) & x = (x_1,\ldots,x_n) \wedge y = (y_1,\ldots,y_n) \\ x \cup y & x = \{x_1,\ldots,x_n\} \wedge y = \{y_1,\ldots,y_n\} \\ x \cup_{\mathrm{bag}} y & x = \{.x_1,\ldots,x_n.\} \wedge y = \{.y_1,\ldots,y_n.\} \\ \emptyset & \text{otherwise} \end{cases}$$

NOTES

— unless otherwise specified by additional constraints (such as a type hierarchy), type interpretations are disjoint, i.e. $I_{\mathbf{Type}}(\tau_1) \cap I_{\mathbf{Type}}(\tau_2) \neq \emptyset \Rightarrow \tau_1 = \tau_2$

— if we know that τ_1 is a subtype of τ_2, we must have $I_{\mathbf{Type}}(\tau_1) \subset I_{\mathbf{Type}}(\tau_2)$

— if we have appropriateness conditions between features and types stating for instance that $\mathrm{app}(\tau, f) = \tau_2$ is the most general allowed type as value of feature f in a feature structure of type τ_1, then we must refine the definition of I with the following constraint:

$$I(\tau[f_i : v_i]) = \{x \in I_{\mathbf{Type}}(\tau) | \forall f_i,\ I(f_i)(x) \in I(v_i,\mu) \cap I_{\mathbf{Type}}(\mathrm{app}(\tau, f_i))\}$$

— one could widen the definition of operation $\oplus$ to allow coercion between the different kinds of collections.

D.5 Satisfiability

A feature structure s is satisfiable with respect to an interpretation I if $I(s) \neq 0$.

A feature structure s is satisfiable if s is satisfiable with respect to some interpretation I.

D.6 Subsumption

A feature structure s subsumes (or generalizes) a feature structure t if $I(s) \supset I(t)$ for all interpretations I.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.7 Unification

Two feature structures, s and t, are unifiable if there exists some feature structure u subsumed by both s and t.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex E
(informative)

Use of feature structures in applications

E.1 Overview

Feature structures are used in increasingly many current grammatical theories, grammar development platforms and natural language processing systems. This annex is intended to help the reader overview the diversity and the usefulness of feature structures in scientific practice.

E.2 Phonological representation

The earliest use of feature structures may be found in phonology. In phonological representation, for example in Chomsky and Halle (1968)[10], the segment of the consonant /s/ is represented in feature structures as follows: {<consonantal +>, <vocalic −>, <voiced −>, <anterior +>, <coronal +>, <continuant +>, <strident +>}

E.3 Grammar formalisms or theories

Almost all the current computational grammar formalisms or theories incorporate feature structures. They use the operation of unification to merge the information encoded in feature structures, hence sometimes the term “unification-based grammars”. LAG is an exception, adopting a strictly time-linear derivation procedure. The following grammar formalisms or theories share a property that they all use feature structures to represent some important aspects of the grammatical information (Shieber, 1986)[48].

a) Generalized Phrase Structure Grammar (GPSG): Gazdar, Klein, Pullum and Sag (1985)[19]

b) Head-driven Phrase Structure Grammar (HPSG): Pollard and Sag (1987; 1994)[43],[44] and Sag, Wasow and Bender (2003)[47]

c) Lexical Functional Grammar (LFG): Bresnan (1982)[5]

d) Functional Unification Grammar (FUG): Kay (1992)[30]

e) Definite Clause Grammar (DCG): Pereira and Warren (1980)[40]

f) Tree Adjoining Grammar (TAG): VijayShanker and Joshi (1988)[55]

g) Left-Associative Grammar (LAG): Hausser (1999)[21]

h) Construction Grammar (CG): Kay (2002)[31]

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

E.4 Computational implementations

In parallel to these grammar formalisms and theories, many computational implementations of feature structure have emerged, some of which are listed below.

a) Attribute Logic Engine (ALE: Carpenter and Penn, 1995)[8]

Created at the Carnegie Mellon University, ALE is one of the oldest systems still being used for the implementation of HPSG grammars. This system is an integrated phrase structure parsing and definite clause logic programming system in which terms are typed feature structures. Originally ALE, based on Kasper-Rounds logic (Kasper and Rounds, 1986) and Carpenter (1992), was created in collaboration between Bob Carpenter and Gerald Penn, but then became the sole work of Gerald Penn and is still being continually upgraded.

b) Advanced Linguistic Engineering Platform (ALEP: Simpkins and Groenendijk, 1994)[49]

Created at the Advanced Information Processing Group at IBM, Belgium, ALEP is a platform for developers of linguistic software and provides a distributed, multitasking (Motif-based) environment containing tools for developing grammars and lexicons. The system comes with a unification-based formalism and parsing, generation and transfer components. A facility to implement two-level morphology is also provided. The ALEP formalism has been developed on the basis of the ET 6.1 project (Alshawi *et al*., 1991). The core of the ET 6.1 formalism follows a rather conservative design and ALEP is very much a traditional rule based grammar development environment. The rules are based on a context free phrase structure backbone with associated types.

c) Abstract-Machine Language (AMALIA: Wintner, Gabrilovich, and Francez, 1997)[55]

AMALIA is a grammar development system that includes a compiler of grammars (for parsing and generation) to abstract-machine instructions, and an interpreter for the abstract-machine language. The generation compiler inverts input grammars (designed for parsing) to a form more suitable for generation. The compiled grammars are then executed by the interpreter using one control strategy, regardless of whether the grammar is the original or the inverted version. They thus obtain a unified, efficient platform for developing reversible grammars.

d) German NLP Processor BABEL (Müller, 1996)[38]

BABEL is a natural-language processing system for German in Prolog. The linguistic theory behind the system is a version of HPSG by Pollard and Sag. It focuses mainly on syntactic issues; word order phenomena in particular. As German is a language with relatively free constituent order, there are problems quite different from those known for the processing of languages like English. BABEL follows the program of linearization grammar begun by Pollard, Levine, and Kasper (1993), Reape (1994), and Kathol (1995).

e) Grammar Development System ConTroll (Götz, 1995)[20]

ConTroll is a grammar development system (or a pure logic program) which supports the implementation of current constraint-based theories. It uses strongly typed feature structures as its principal data structure and offers definite relations, universal constraints, and lexical rules to express grammar constraints. ConTroll comes with the graphical interface Xtroll which allows displaying of AVMs and trees, as well as a graphical debugger. The ConTroll-System was based on the logical foundations of HPSG grammars created by Paul King with his Speciate Re-entrant Logic (SRL).

f) Typed Feature Structure Description Langauge CL-ONE (Manandhar, 1994)[37]

CL-ONE extends ProFIT's typed feature structure description language. The system comprises constraint solvers for set descriptions, linear precedence and guarded constraints. Set and linear precedence constraints are defined over constraint terms (cterm), which are internal CL-ONE data structures. Unification of sets differs from standard Prolog terms unification. The system has been designed in the

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

project The Reusability of Grammatical Resources, at the University of Edinburgh and at Universität des Saarlandes, Saarbrücken.

g) Comprehensive Unification Formalism (CUF: Dörre and Dorna, 1993)[14]

CUF has been developed at IMS, University of Stuttgart, within the DYANA research project as an implementational tool for grammar development. CUF has been designed to provide mechanisms broadly used in current unification based grammar formalisms such as HPSG or LFG. The main features of CUF are a very general type system and relational dependencies on feature structures. It is a constraint based system, with no specialized components, e.g., for morphology or transfer. It can be seen as a general constraint solver and it does not include a parser or a generator.

h) Formal Language DATR (Evans and Gazdar, 1996)[18]

Developed at the University of Sussex, DATR is a formal language for representing a restricted class of inheritance networks, permitting both multiple and default inheritance with path/value equations. DATR has been designed specifically for lexical knowledge representation. QDATR is an implementation of the DATR formalism. It supports syntactic analysis, text generation, machine translation and can be used for testing linguistic theories making use of non-monotonic inheritances.

i) Database Semantics (DBS: Hausser, 1999, 2001)[21],[22]

DBS stands for Database Semantics. Developed by Roland R. Hausser at the University of Erlangen, it has been implemented with JAVA to model natural language communication in the form of an artificial agent (talking robot). In DBS, content stored in the context and coded in language has the form of concatenated propositions. Propositional content is defined as a set of features structures, called *proplets*. Dealing with various English constructions, the system demonstrates how natural language communication is successfully carried out at both the hearer and speaker modes by various processes of navigation through a semantic database. Based on LAG, navigation proceeds in a strictly time-linear order, while operating on feature structures without unification.

j) Grammar Writer's Workbench for LFG (Kaplan and Maxwell, 1996)[27]

The Xerox LFG Grammar Writer's Workbench is a complete parsing implementation of the LFG syntactic formalism, including various features introduced since the original Kaplan and Bresnan (1982)[26] paper (functional uncertainty, functional precedence, generalization for coordination, multiple projections, etc.). It includes a very rich c-structure rule notation, plus various kinds of abbreviatory devices (parameterized templates, macros, etc.). It does not directly implement recent proposals for lexical mapping theory, although templates can be used to simulate some of its effects. The system has an elaborate mouse-driven interface for displaying various grammatical structures and substructures – the idea is to help a linguist understand and debug a grammar without having to comprehend the details of specific processing algorithms. The workbench runs on most Unix systems (Sun, DEC, HP, …) and under DOS on PC's, although most of our experience is on Sun's. It does not have a teletype interface – it only runs as a graphical program. It requires at least 16MB of ram on UNIX and 8MB under DOS, plus 40 or more MB of disk (for program storage and swapping).

k) Graph Unification and Logic Programming (GULP: Covington, 1989)[13]

GULP is a preprocessor for handling feature structures in Prolog programs. It solves a pesky problem with Prolog, i.e. the lack of a good way to represent feature structures in which features are identified by name rather than position.

l) Programming Language LIFE (Aït-Kaci and Podelski, 1993)[2]

LIFE is an experimental programming language proposing to integrate three orthogonal programming paradigms proven useful for symbolic computation. From the programmer's standpoint, it may be perceived as a language supporting logic programming, functional programming, and object-oriented programming. From a formal perspective, it may be seen as an instance (or rather a composition of three instances) of a Constraint Logic Programming scheme due to Hohfeld and Smolka (1988)[23] refining that

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

of Jaffar and Lassez (1986). LIFE's object unification is seen as constraint-solving over specific domains. LIFE is built on work by Smolka and Rounds to develop type-theoretic, logical, and algebraic renditions of a calculus of order-sorted feature approximations.

m) Programming Environment LiLFeS (Makino, *et al*., 2000)[36]

The LiLFeS system provides a programming environment with efficient processing of typed feature structures. It is a Prolog-like logic-programming language integrating feature-structure descriptions and definite-clause programs by expressing predicate arguments of Prolog in typed feature structures instead of first-order terms. A user who already knows Prolog syntax can easily understand the LiLFeS syntax. The core engine of the LiLFeS system is developed in C++ as an implementation of Warren's Abstract Machine (Aït-Kaci, 1991) and Abstract Machine for Attribute-Value Logics (Carpenter and Qu, 1995) so as to maximize efficiency.

Large-scale systems such as grammars, parsers and grammar-development tools have been developed in LiLFeS. For example, a wide-coverage treebank grammar of HPSG for analyzing both syntactic structures and predicate-argument relations is developed by a grammar-development tool implemented in LiLFeS (Miyao, *et al*., 2004). The LiLFeS system also provides a C++ class library for manipulating typed feature structures and their databases. Enju (Tsuruoka, *et al*., 2004), an efficient probabilistic HPSG parser for the English treebank grammar, is implemented in C++ using this library.

n) Linguistic Knowledge Biliding (LKB: Copestake, 2002) [12]

The Linguistic Knowledge Building (LKB) system is a grammar and lexicon development environment for use with constraint-based linguistic formalisms. The LKB software is distributed by the LinGO initiative, a loosely-organized consortium of research groups working on unification-based grammars and processing schemes.

o) Left-Associative Grammar Tool with Attributes (MALAGA: Beutel, 1997)[4]

MALAGA is a development environment for Left-Associative Grammars (LAGs) with Atttribute-value structures. LAG is a formalism that describes the analysis and generation of words and sentences strictly from the left to the right. It has been introduced by Roland Hausser. The Grammars for morphological and/or syntactical analysis are written in a Pascalish language, but it uses a powerful dynamic typing system with attribute-value structures and lists that can be nested with a collection of newly-defined operators and standard functions. It contains constructs to branch the program execution into parallel paths when a grammatical ambiguity is encountered. The grammars are compiled into virtual code that is executed by an interpreter. The toolkit comprises parser generators for the development of morphological and syntactic parsers (including sample grammars), source-code lexicon/grammar debuggers (with Emacs modes), and an optional visualization tool to display derivation paths and categorial values using GTK+. The toolkit is written in ANSI/ISO-compliant C and its source code is freely available. It can be used and distributed under the terms of the GNU General Public License. It should work out-of-the-box on most POSIX systems. Porting to other platforms should be easy. A Windows port has already been made publicly available.

p) Platform for Advanced Grammar Engineering (PAGE: Krieger and Schäfer, 1994a; 1994b)[32],[33]

PAGE is a grammar development environment and runtime system which evolved from the DISCO project (Uszkoreit, *et al*., 1994) at the German Research Center for Artificial Intelligence (DFKI). The system has been designed to facilitate development of grammatical resources based on typed feature logics. It consists of several specialized modules which provide mechanisms that can be used to directly implement notions of HPSG, LFG and other grammar formalisms. PAGE provides a general type system (TDL), a feature constraint solver (UDiNe), a chart parser, a feature structure editor (Fegramed), a chart display to visualize type hierarchies and an ASCII-based command shell. The remainder will concentrate mostly on the TDL module which is the best documented and constitutes the fundamental part of the environment. The description of TDL is based on Krieger and Schäfer (1994a; 1994b)[32],[33].

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

q) PET (Callmeier, 2000)[6]

The PET system for efficient processing of unification-based grammars is being developed at the Department of Computational Linguistics at Saarland University by Ulrich Callmeier and others. The system is developed in C so as to maximize efficiency, and employs a quick check technique for unifiability. It is the most efficient parser in the benchmarks of parsers for the LinGO grammar in 1999 (Oepen, *et al*., 2000).

r) Prolog with Features, Inheritance and Templates (ProFIT: Erbach, 1994)[17]

ProFIT is an extension of Prolog, developed at Universität des Saarlandes, Saarbrücken. Its programs consist of data type declarations and Prolog definite clauses. ProFIT provides mechanisms to declare an inheritance hierarchy and define feature structures. Templates are widely used to simplify descriptions, encode constraint-like conditions and abstract over complex data structures.

s) Type Description Language (TDL: Krieger and Schäfer, 1994a; 1994b)[32],[33]

TDL is a typed feature-based language, which is specifically designed to support highly lexicalized grammar theories, such as HPSG,FUG, or CUG. TDL offers the possibility to define (possibly recursive) types, consisting of type and feature constraints over the Boolean connectives AND, OR, and NOT, where the types are arranged in a subsumption hierarchy. TDL distinguishes between AVM types (open-world reasoning) and sort types (closed-world reasoning) and allows the declaration of partitions and incompatible types. Working with partially and fully expanded types as well as with undefined types is possible, both at definition and at run time. TDL is incremental in that it allows the redefinition of types. Efficient reasoning is accomplished through specialized modules. Large grammars and lexicons for English, German, and Japanese are available.

t) Typed Feature Structure Representation Formalism (TFS: Emele and Zajac, 1990; Emele, 1993)[16],[15]

TFS was developed in the German Polygloss project at IMS, University of Stuttgart. The inheritance-based constraint architecture embodied in the TFS system integrates two computational paradigms: the object-oriented approach offers complex, recursive, possibly nested, record objects represented as typed feature structures with attribute-value restrictions and (in)equality constraints, and multiple inheritance; the relational programming approach offers declarativity, logical variables, non-determinism with backtracking, and existential query evaluation. The constraint-based properties of the TFS formalism are fully exploited in the Delis lexicon model and pertaining tools. These include corpus search and support for the major steps and types of activity in lexicon building: creation, population and modification of lexical models; exportation towards formats for both NLP and human use.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] AÏT-KACI, HASSAN. *Warren's Abstract Machine, A Tutorial Reconstruction*. The MIT Press, 1991

[2] AÏT-KACI, HASSAN and PODELSKI, ANDREAS. Towards a meaning of LIFE. *Journal of Logic Programming*, **16**, 1993, pp. 195-234

[3] ALSHAWI, HIYAN, CARTER, DAVID, GAMBACK, BJORN, and RAYNER, MANNY. Translation by quasi-logical form transfer. In: *Proceedings 29th Annual Meeting of Association of Computational Linguistics*, Berkeley, CA, 1991

[4] BEUTEL, BJÖRN. *MALAGA 4.0 On-line documentation*, Abteilung für Linguistische Informatik, Friedrich Alexander Universität, Erlangen Nürnberg, Germany, 1997

[5] BRESNAN, JOAN W., editor. *The Mental Representation of Grammatical Relations*. The MIT Press, Cambridge, Massachusetts, 1982

[6] CALLMEIER, ULRICH. Preprocessing and encoding techniques in PET. In: *Collaborative Language Engineering: A Case Study in Efficient Grammar-based Processing*. (TSUJII, Junichi, OEPEN, Stephan, FLICKINGE, Dan and USZKOREIT, Hans, eds.). CSLI Publications, 2000, pp. 127-143

[7] CARPENTER, BOB. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, 1992

[8] CARPENTER, BOB and PENN, GERARD. Compiling typed attribute-value logic grammars. In: *Current Issues in Parsing Technologies*, **2**. (BUNT, H. and TOMITA, M. eds.). Kluwer, 1995

[9] CARPENTER, BOB and QU, YAN. An abstract machine for attribute-value logics. In: *Proceedings of IWPT95*, 1995, pp. 59-70

[10] CHOMSKY, NOAM and HALLE, MORRIS. *The Sound Pattern of English*. Harper & Row, New York, 1968

[11] VILLEMONTE DE LA CLERGERIE, ERIC. DyALog : A Tabular Logic Programming based enviroment for NLP. In: *Preliminary Proceedings of the Second International Workshop on Constraint Solving and Lanuage Processing*, 2005

[12] COPESTAKE, ANN. *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford, 2002

[13] COVINGTON, MICHAEL A. GULP 2.0: *An Extension of Prolog for Unification-Based Grammar*. Research Report AI-1989-01, Artificial Intelligence Program, University of Georgia, 1989

[14] DÖRRE, JOCHEN and DORNA, MICHAEL. CUF: A formalism for linguistic knowledge representation. In: *Computational Aspects of Constraint-based Linguistic Descriptions I*, DYANA2 Deliverable R1.2.A. (DÖRRE, Jochen ed.). Universität Stuttgart, 1993, pp. 1-22

[15] EMELE, MARTIN. TFS — the typed feature structure representation formalism. In: *Proceedings of the EAGLES Workshop on Implemented Formalisms*. DFKI. (USZKOREIT, Hans ed.). 1993

[16] EMELE, MARTIN and ZAJAC, RÉMI. Typed unification grammars. In: *Proceedings of the 13th International Conference on Computational Linguistics*, 1990

[17] ERBACH, GREGOR. ProFit — prolog with features, inheritance, and templates. CLAUS Report no. 42, Saarbrücken: Universität des Saarlandes, 1994

[18] EVANS, ROGER and GAZDAR, GERALD. DATR: A language for lexical knowledge representation. *Computational Linguistics*, **22**, 1996, pp. 167–216

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[19] PULLUM, GEOFFREY, GAZDAR, GERALD, KLEIN, EWAN and SAG, IVAN. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, MA, 1985

[20] GÖTZ, THILO and MEURERS, WALT DETMAR. Compiling HPSG type constraints into definite clause programs. In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 1995

[21] HAUSSER, ROLAND. *Foundations of Computational Linguistics: Human-Computer Communication in Natural Language*. Springer-Verlag, Berlin, 2nd edition, 1999

[22] HAUSSER, ROLAND. Database semantics for natural language. *Artificial Intelligence*, **130**(1), 2001, pp. 27–74

[23] HOHFELD, MARKUS and SMOLKA, GERT. *Definite relations over constraint languages*. LILOG report 53, IWBS, IBM Deutschland, 1988

[24] JAFFAR, JOXAN and LASSEZ, JEAN-LOUIS. Constraint logic programming. In: *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of programming languages*, pp. 111–119, Munich, 1987

[25] JOHNSON, MARK. *Attribute-Value Logic and the Theory of Grammar*. CSLI, Stanford, 1988. CSLI Lecture Notes 16

[26] KAPLAN, RONALD M. and BRESNAN, JOAN W. Lexical-functional grammar: A formal system for grammatical representation. In: *The Mental Representation of Grammatical Relations*. (Bresnan, Joan W. ed.). The MIT Press, 1982, pp. 173-281

[27] KAPLAN, RONALD M. and MAXWELL III, JOHN T. *Grammar Writer's Workbench*. Xerox Corporation, 1996. ftp://ftp.parc.xerox.com/pub/lfg/lfgmanual.ps

[28] KASPER, ROBERT T. and ROUNDS, WILLIAM C. A logical semantics for feature structures. In: *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, 1986, pp. 235-242

[29] KATHOL, ANDREAS. *Linearization-Based German Syntax*. PhD thesis, Ohio State University, 1995

[30] KAY, MARTIN. Unification. In: *Computational Linguistics and Formal Semantics. (*ROSNER, Michael and JOHNSON, Roderick, eds.). Cambridge University Press, Cambridge, 1992. pp. 1-30

[31] KAY, PAUL. An informal sketch of a formal architecture for construction grammar. *Grammars*, **5**, 2002, p. 1-19

[32] KRIEGER, HANS-ULRICH and SCHÄFER, ULRICH. *TDL — a type description language for HPSG, part 1: Overview*. Technical report RR9437, DFKI, 11 1994

[33] KRIEGER, HANS-ULRICH and SCHÄFER, ULRICH. *TDL — a type description language for HPSG, part 2: User guide*. Technical report d9414, DFKI, 11 1994

[34] LANGENDOEN, TERENCE D. and SIMONS, GARY F. A rationale for the TEI recommendations for feature structure markup. *Computers and the Humanities*, **29**, 1995, pp. 191–209

[35] LEE, Kiyong, BURNARD, Lou, ROMARY, Laurent, DE LA CLERGERIE, Eric, SCHAEFER, Ulrich, DECLERCK, Theirry, BAUMAN, Syd, BUNT, Harry, CLÉMENT, Lionel, ERJAVEC, Tomaz , ROUSSANALY, Azim, and ROUX, Claude. Towards an international standard on feature structure representation (2). *Workshop on a Registry of Linguistic Data Categories within an Integrated Language Resources Repository Area (INTERA)*, LREC 2004 International Conference, Lisbon, 2004

[36] MAKINO, TAKAKI, MIYAO, Yusuke, TORISAWA, KENTARO and TSUJII, JUN'ICHI. Native-code compilation of feature structures. In: *Collaborative Language Engineering: A Case Study in Efficient Grammar-based*

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Processing. (TSUJII, Junichi, OEPEN, Stephan, FLICKINGER, Dan and USZKOREIT, Hans eds.). CSLI Publications, 2000

[37] MANANDHAR, SURESH. *User's Guxml:ide for CLONE*. Centre for Cognitive Science, University of Edinburgh, Scotland, 1994

[38] MÜLLER, STEFAN. The babel system – an HPSG prolog implementation. In: *Proceedings of the Fourth International Conference on the Practical Application of Prolog*, 1996, pp. 263-277, London. http://www.cl.uni-bremen.de/~stefan/Pub/babel.html

[39] PEREIRA, FERNANDO C.N. *Grammars and logics of partial information*. SRI International Technical Note 420, SRI International, Menlo Park, CA, 1987

[40] PEREIRA, FERNANDO C.N. and WARREN, DAVID H.D. Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, **13**, 1980, pp. 231-278

[41] POLLARD, CARL, LEVINE, ROBERT, and KASPER, ROBERT. *Studies in constituent ordering: Toward a theory of linearization in headdriven phrase structure grammar*, 1994. Grant proposal to the National Science Foundation

[42] POLLARD, CARL J. and MOSHIER, M. ANDREW. Unifying partial descriptions of sets. In: *Information, Language and Cognition*. (Hanson, Philip P. ed.). The University of British Columbia Press, Vancouver, 1990, pp. 285-322

[43] POLLARD, CARL J. and SAG, IVAN A. *Fundamentals*, volume 1 of Information-based Syntax and Semantics. CSLI, Stanford, 1987. CSLI Lecture Notes 13

[44] POLLARD, CARL J. and SAG, IVAN A. *Head-driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, 1994

[45] REAPE, MIKE. Domain union and word order variation in German. In: *German in Head-Driven Phrase Structure Grammar.* (J. Nerbonne *et al*., eds.). CSLI Publications, 1994, pp. 151-198

[46] SAG, IVAN A. and WASOW, THOMAS. *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, 1999

[47] SAG, IVAN A., WASOW, THOMAS and BENDER, EMILY M. *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, 2nd edition, 2003

[48] SHIEBER, STUART M. *An Introduction to Unification-Based Approaches to Grammar*. CSLI, Stanford, 1986. CSLI Lecture Notes 4

[49] SIMPKINS, NEIL and GROENENDIJK, MARIUS. *The ALEP Project*. Technical report, Cray Systems/CEC, Luxembourg, 1994

[50] SPERBERG-MCQUEEN, C.M. and BURNARD, LOU, eds. *Guidelines for Text Encoding and Interchange: TEI P5*. Text Encoding Initiative Consortium, 2004. Available at http://www.tei-c.org/P5/

[51] TSUJII, JUN'ICHI, OEPEN, STEPHAN, FLICKINGER, DAN and USZKOREIT, HANS. *Collaborative Language Engineering: A Case Study in Efficient Grammar-based Processing*. CSLI Publications, 2000

[52] SPERBERG-MCQUEEN, C.M., BRAY, TIM, PAOLI, JEAN and MALEr, Eve (eds.). *eXtensible Markup Language (XML) 1.0*. second edition, 2000. World Wide Web Consortium (W3C) Recommendation. Referenced in ISO 16642 and available at http://www.w3.org/TR/2000/REC-xml-20001006

[53] TSURUOKA, YOSHIMASA, MIYAO, YUSUKE and TSUJII, JUN'ICHI. Towards efficient probabilistic HPSG parsing: integrating semantic and syntactic preference to guide the parsing. In: *Proceedings of*

© ISO 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IJCNLP04 Workshop: Beyond Shallow Analyses Formalisms and Statistical Modeling for Deep Analyses, 2004

[54] VIJAYSHANKER, K. and JOSHI, ARAVIND K. Feature-structure based tree adjoining grammar. In: *Proceedings of 12nd International Conference on Computational Linguistics(COLING'88)*, 1988

[55] WINTNER, SHULY, GABRILOVICH, EVGENIY and FRANCEZ, NISSIM. Amalia — a unified platform for parsing and generation. In: *Proceedings of Recent Advances in Natural Language Programming (RANLP97)*, 1997, pp. 135-142,

[56] NINOMIYA, TAKASHI, MIYAO, YUSUKE and TSUJII, JUN'ICHI. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the penn treebank. In: *Proceedings of IJCNLP04*, 2004

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

ICS 01.140.20

Price based on 78 pages

© ISO 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale