Contents

ODD SUBSET

1 Feature Structures

A feature structure is a general purpose data structure which identifies and groups together individual features, each of which associates a name with one or more values. Because of the generality of feature structures, they can be used to represent many different kinds of information, but they are of particular usefulness in the representation of linguistic analyses, especially where such analyses are partial, or underspecified. Feature structures represent the interrelations among various pieces of information, and their instantiation in markup provides a metalanguage for the generic representation of analyses and interpretations. Moreover, this instantiation allows feature values to be of specific types, and for restrictions to be placed on the values for particular features, by means of feature system declarations.

1.1 Organization of this Chapter

This chapter is organized as follows. Following this introduction, section *1.1.2. Elementary Feature Structures and the Binary Feature Value* introduces the elements `<fs>` and `<f>`, used to represent feature structures and features respectively, together with the elementary binary feature value. Section *1.1.3. Other Atomic Feature Values* introduces elements for representing other kinds of atomic feature values such as symbolic, numeric, and string values. Section *1.1.4. Feature and Feature-Value Libraries* introduces the notion of predefined libraries or groups of features or feature values along with methods for referencing their components. Section *1.1.5. Feature Structures as Complex Feature Values* introduces complex values, in particular feature-structures as values, thus enabling feature structures to be recursively defined. Section *1.1.7. Collections as Complex Feature Values* discusses other complex values, in particular values which are collections, organized as sets, bags, and lists. Section *1.1.8. Feature Value Expressions* discusses how the operations of alternation, negation, and collection of feature values may be represented. Section *1.1.9. Default and Uncertain Values* discusses ways of representing underspecified, default, or uncertain values. Section *1.1.10. Linking Text and Analysis* discusses how analyses may be linked to other parts of an encoded text. Formal definitions for all the elements introduced in this chapter are provided in section *1.1.11. Formal Definition and Implementation*.

1.2 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 it may be complex, that is, it may itself be a feature structure, thus providing a degree of recursion. Values may be under-specified or defaulted in various ways. These possibilities are all described in more detail in this and the following sections.

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:

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

  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. Selected attributes:

name provides a name for the feature.

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

- **<binary>** represents the value for a feature-value specification which can take either of a pair of values.

  value supplies a Boolean value (true or false, plus or minus).

The attributes *feats* and the *fVal* are not discussed in this section: they provide an alternative way of indicating the content of an element, as further discussed in section *1.1.4. 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].[1]

Example 1.1

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

This representation may be encoded in XML as follows:

Example 1.2

```
<fs type="phonological segment">
    <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>
```

Note that `<fs>` elements may have an optional *type* attribute to indicate the kind of feature structure in question, whereas `<f>` elements must have a *name* attribute to indicate the name of the feature. Feature structures need not be typed, but features must be named. Similarly, the `<fs>` element may be empty, but the `<f>` element must 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 discussed in «chapter FD of the TEI Guidelines».

Although we have used the term binary 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 *1.1.11. Formal Definition and Implementation*.

1.3 Other Atomic Feature Values

Features may take other kinds of atomic value. In this section, we define elements which may be used to represent: symbolic values, numeric values, and string values. The module defined by this chapter allows for the specification of additional datatypes if necessary, by extending the underlying class class.singleValue. 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.

- **<symbol>** provides a symbolic feature value. Selected attributes:

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

[1] Adapted from Noam Chomsky and Morris Halle, The Sound Pattern of English. New York: Harper & Row, 1968, p. 415.

- **<numeric>** represents a numeric value or range of values for a feature.

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

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:

Example 1.3

```
<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:

Example 1.4

```
<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:

Example 1.5

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

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:

Example 1.6

```
<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:

Example 1.7

```
<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 mm is a feature of interest for some address, one might represent this by an encoding like the following:

Example 1.8

```
<fs>
 <f name="dailyRainFall">
  <numeric value="0.0" max="1.3" trunc="no"/>
 </f>
</fs>
```

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

Example 1.9

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

represents only two possible values: 0 and 1.

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 then 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 3 2) is required to enumerate all the possible combinations of individual case, gender and number values in the preceding illustration. We discuss the use of such libraries and their representation in XML further in section *1.1.4. Feature and Feature-Value Libraries* 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 discussed in «chapter FD of the TEI Guidelines».

1.4 Feature and Feature-Value Libraries

As the examples in the preceding section 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 re-used 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 section. The following elements are used for representing feature, and feature-value libraries:

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

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

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

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

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

Example 1.10

```
<fLib type="phonological features">
    <f id="CNS1" name="consonantal"> <binary value="true"/> </f>
    <f id="CNS0" name="consonantal"> <binary value="false"/> </f>
    <f id="VOC1" name="vocalic">     <binary value="true"/> </f>
    <f id="VOC0" name="vocalic">     <binary value="false"/> </f>
    <f id="VOI1" name="voiced">      <binary value="true"/> </f>
    <f id="VOI0" name="voiced">      <binary value="false"/> </f>
    <f id="ANT1" name="anterior">    <binary value="true"/> </f>
    <f id="ANT0" name="anterior">    <binary value="false"/> </f>
    <f id="COR1" name="coronal">     <binary value="true"/> </f>
    <f id="COR0" name="coronal">     <binary value="false"/> </f>
    <f id="CNT1" name="continuant">  <binary value="true"/> </f>
    <f id="CNT0" name="continuant">  <binary value="false"/> </f>
    <f id="STR1" name="strident">    <binary value="true"/> </f>
    <f 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.

Example 1.11

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

The preceding are but four of the 128 logically possible fully specified phonological segments using the seven binary features listed in the feature library. Presumably not all combinations of features correspond to phonological segments (there are no strident vowels, for example). The legal combinations, however, can be collected together, each one represented as an identifiable <fs> element within a feature-value library, as in the following example:

Example 1.12

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

Once defined, these feature structure values can also be reused. Other <fs> elements may invoke them by reference, using the fVal attribute; for example, one might use them to define a feature structure such as :

Example 1.13

```
 <fs type="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 (for example, using the TEI global ana attribute), or by means of standoff annotation techniques (for example, using the TEI <link> element): see further section *1.1.10. Linking Text and Analysis* below.

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 <var> element, as discussed in *1.1.6. Re-entrant feature structures* below.

1.5 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 a <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 a <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:

Example 1.14

```
<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="rel"><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 we maintain a feature-value library for each of the major syntactic categories (N, V, ADJ, PREP):

Example 1.15

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

 <fs 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:

Example 1.16

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

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

Example 1.17

```
<fLib type="categorial features">
    <f id="N1" name="nominal"><binary value="true"/></f>
    <f id="N0" name="nominal"><binary value="false"/></f>
    <f id="V1" name="verbal"><binary value="true"/></f>
    <f 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:

Example 1.18

```
<fs type="word">
 <f name="surf"><stringVal>love</stringVal></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 recommended only for cases where the feature value required is a reference to some predefined feature structure.

1.6 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 `<var>` element is provided as a means of labelling each such re-entrancy point:

- **<var>** indicates that the feature value is shared with other values bearing the same label within the current scope.

  label supplies a label for the variable.

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:

Example 1.19

```
<fs id="NVA">
 <f name="nominal">
   <fs>
     <f name="nm-num">
      <var label="L1">
        <symbol value="singular"/>
     </var>
    </f>
    <!-- other nominal features -->
   </fs>
 </f>

 <f name="verbal">
   <fs>
     <f name="vb-num">
       <var label="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 `var` elements, are understood to be references to the same point in the feature structure, which is labelled by their *label* attribute.

The scope of the labels used for 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) any labels 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, for example in this way:

Example 1.20

```
<fs fVal="NVA"/>
```

then the labels in the example would be interpreted as if they had the value NVAL1.

1.7 Collections as Complex Feature Values

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

- **<coll>** contains (or points to) several feature structures which together provide a value for their parent feature element, organized as indicated by the value of the org attribute.

  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.

  fVal pointer to features.

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 must 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 <coll> defines a list of values. If the <coll> element is empty, the assumption is that it represents the null list, set, or bag, unless the *feats* attribute is used to specify its contents. If values are supplied within a <coll> element which also specifies values on its *feats* attribute, the implication is that the two sets of values are to be unified.

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:

Example 1.21

```
<fs id="p027" type="person">
    <f name="forenames">
      <coll>
        <f name="name"><stringVal>Daniel</stringVal></f>
        <f name="name"><stringVal>Edouard</stringVal></f>
      </coll>
    </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">
        <coll org="set" feats="pnb005 pfb010 prb001"/>
    </f>
 </fs>
```

In this example, the <coll> 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 we assume are held in some external feature value library. The <coll> element is used a second time to indicate that the persons's siblings should be regarded as constituting a set rather than a list.

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 <coll> 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.

Example 1.22

```
<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:

Example 1.23

```
<fs type="word">
   <f name="word.oddss"> <symbol value="verb"/> </f>
   <f name="tense"> <symbol value="present"/> </f>
   <f name="agreement">
      <coll org="set">
         <symbol value="third"/>
         <symbol value="singular"/>
      </coll>
   </f>
 </fs>
```

The <coll> 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:

Example 1.24

```
<fs>
       <f name="lex">
         <symbol value="auxquels"/>
       </f>
       <f name="maf">
         <coll org="list">
           <fs>
             <f name="cat"><symbol value="prep"/></f>
           </fs>
           <fs>
             <f name="cat"><symbol value="pronoun"/></f>
             <f name="kind"><symbol value="rel"/></f>
             <f name="num"><symbol value="pl"/></f>
             <f name="gender"><symbol value="masc"/></f>
           </fs>
         </coll>
       </f>
     </fs>
```

The set, bag or list which has no members is known as the null (or empty) set, bag or list. A <coll> 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 (above) had no siblings, we might specify the 'siblings' feature as follows.

Example 1.25

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

A <coll> element may also collect together one or more other <coll> 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 <coll> elements: if it is desired to produce the union of two sets, the <vColl> element discussed below should be used to make a new collection from the two sets.

1.8 Feature Value Expressions

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:

- **<vAlt>** represents a feature value which is the disjunction of the values it contains.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<vNot>** represents a feature value which is the negation of its content.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<vColl>** represents a feature value which is the result of collecting together the feature values it combines, using the organization specified by the ORG attribute.

  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.

1.8.1 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 we are using a feature system to describe residential property, using such features as 'number.of.bathrooms'. In a particular case, we might wish to represent uncertainty as to whether a house has two or three bathrooms. As we have already shown, one simple way to represent this would be with a numeric maximum:

Example 1.26

```
<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:

Example 1.27

```
<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 must be represented as a feature 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:

Example 1.28

```
<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 <coll> element may be included in the alternation as follows:

Example 1.29

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

Example 1.30

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

Example 1.31

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

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

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 stand-off technique, for example using the general-purpose <alt> element discussed in TEI P5, be used, rather than the special-purpose <vAlt> element.

1.8.2 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:

Example 1.32

```
<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 FSD or by some other means. Then the following two specifications are equivalent:

Example 1.33

```
(i) <f name="case">
      <vNot><symbol value="genitive"/></vNot></f>

(ii) <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 a <coll> elements) or feature structures (represented by <fs> elements). The negation of any complex value is understood to be the set of values of the same type which cannot be unified by 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 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.

1.8.3 Collection of values

The <vColl> 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.

As an example, suppose that we wish 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 <coll> element as in the following example:

Example 1.34

```
<fs>
 <f name="genders">
  <coll org="set">
```

```
    <sym value="masculine"/>
    <sym value="neuter"/>
    <sym value="feminine"/>
   </coll>
  </f>
 </fs>
```

Suppose however that we discover for some language 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 <vColl> element can be used to achieve this:

Example 1.35

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

1.9 Default and Uncertain 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, this information is conveyed by the feature system declaration. Using this, or some other system, we 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 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 special element is provided for this purpose:

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

The value of an empty <f> element which also lacks a 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 are equivalent.

Example 1.36

```
<f name="gender"/>
<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:

Example 1.37

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

Example 1.38

```
<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:

Example 1.39

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

Example 1.40

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

1.10 Linking Text and Analysis

Text elements can be linked with feature structures using any of the linking methods discussed elsewhere in the Guidelines (see for example sections «AIATTS» and «AILA»). . 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:

Example 1.41

```
<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, we cite here only the structures needed for the first six words of the sample sentence):

Example 1.42

```
<fsLib id="C6" type="Claws 6 POS Codes">
    <!-- ... -->
    <fs id="ajs" type="grammatical structure" feats="wj ds"/>
    <fs id="at0" type="grammatical structure" feats="wl"/>
    <fs id="pnp" type="grammatical structure" feats="wr rp"/>
    <fs id="vvd" type="grammatical structure" feats="wv bv fd"/>
    <fs id="prp" type="grammatical structure" feats="wp bp"/>
    <fs id="nn1" type="grammatical structure" 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):

Example 1.43

```
<fLib>
    <!-- ... -->
    <f id="bv" name="verbbase"> <sym value="main"/> </f>
    <f id="bp" name="prepbase"> <sym value="lexical"/> </f>
    <f id="ds" name="degree">   <sym value="superlative"/> </f>
    <f id="fd" name="verbform"> <sym value="ed"/> </f>
    <f id="ns" name="number">   <sym value="singular"/> </f>
    <f id="rp" name="prontype"> <sym value="personal"/> </f>
    <f id="tc" name="nountype"> <sym value="common"/> </f>
    <f id="wj" name="class">    <sym value="adjective"/> </f>
    <f id="wl" name="class">    <sym value="article"/> </f>
    <f id="wn" name="class">    <sym value="noun"/> </f>
    <f id="wp" name="class">    <sym value="preposition"/> </f>
    <f id="wr" name="class">    <sym value="pronoun"/> </f>
    <f id="wv" name="class">    <sym value="verb"/> </f>
    <!-- ... -->
 </fLib>
```

Alternatively, a stand-off 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.

Example 1.44

```
<text>
   <!-- ... -->
   <body>
 <!-- ... -->
  <s id="S1">
   <w id="S1W1"><c id="S1W1C1">C</c>ae<c id="S1W1C2">s</c>ar</w>
   <w id="S1W2"><c id="S1W2C1">s</c>ei<c id="S1W2C2">z</c>e<c id="S1W2C3">d</c></w>
   <w id="S1W3">con<c id="S1W3C1">t</c>rol</w>.
 </s>
 <!-- ... -->
  </body>
   <fvLib 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 stand-off solution requires that every component to be linked to must bear an identifier. To handle the POS tagging example above, therefore, each annotated element would need an identifier of some sort, as follows:

Example 1.45

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

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

Example 1.46

```
<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>
```

1.11 Formal Definition and Implementation

This elements discussed in this chapter constitute a module of the TEI scheme which is formally defined as follows:

Module Feature Structures:

In a TEI conformant document, this module is selected as described in TEI P5, chapter ST

2 Class catalogue

3 Pattern catalogue

4 Element catalogue

4.1 binary [element]

Description: (binary value) represents the value for a feature-value specification which can take either of a pair of values.

Classes: tei.singleVal

Declaration:

```
element binary { tei.global.attributes, binary.attributes.value, empty }
```

Attributes: (In addition to global attributes and those inherited from tei.singleVal)

value supplies a Boolean value (true or false, plus or minus).

Example:

```
<f name="strident">
 <binary value="true"/>
</f>
<f name="exclusive">
 <binary value="no"/>
</f>
```

The value attribute may take any value permitted for attributes of the datatype Boolean: this includes for example the strings `true`, `yes`, or `1` which are all equivalent.

Module: iso-fs

4.2 coll [element]

Description: (collection of values) contains (or points to) several feature structures which together provide a value for their parent feature element, organized as indicated by the value of the org attribute.

Classes: tei.complexVal

Declaration:

```
element coll
{
   tei.global.attributes,
   coll.attributes.org,
   coll.attributes.fVal,
   ( ( fs | tei.singleVal )* )
}
```

Attributes: (In addition to global attributes and those inherited from tei.complexVal)

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.

fVal pointer to features.

Example:

```
<f name="foo">
 <coll org="set" feats="f1 f2 f3"/>
</f>
```

This and the next example are equivalent in meaning.

Example:

```
<f name="foo">
 <coll>
  <fs id="f1"><!-- ... -->
  </fs>
  <fs id="f2"><!-- ... -->
  </fs>
  <fs id="f3"><!-- ... -->
  </fs>
 </coll>
</f>
```

Example:

```
<fs>
 <f name="lex">
  <symbol value="auxquels"/>
 </f>
 <f name="maf">
  <coll org="list">
   <fs>
    <f name="cat">
     <symbol value="prep"/>
    </f>
   </fs>
   <fs>
    <f name="cat">
     <symbol value="pronoun"/>
    </f>
    <f name="kind">
     <symbol value="rel"/>
    </f>
    <f name="num">
     <symbol value="pl"/>
    </f>
    <f name="gender">
     <symbol value="masc"/>
    </f>
   </fs>
  </coll>
 </f>
</fs>
```

Module: iso-fs

4.3 default [element]

Description: (Default feature value) provides default value for a feature.

Attributes: Global attributes only

Classes: tei.singleVal

Declaration:

```
element default { tei.global.attributes, empty }
```

Example:

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

Module: iso-fs

4.4 f [element]

Description: (Feature) represents a feature value specification, that is, the association of a name with a value of any of several different types.

Declaration:

```
element f
{
   tei.global.attributes,
   f.attributes.name,
   f.attributes.fVal,
   tei.featureVal*
}
```

Attributes: (In addition to global attributes)

name provides a name for the feature.

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

Example:

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

If the element is empty then a value must be supplied for the *fVal* attribute.

Module: iso-fs

4.5 fLib [element]

Description: (Feature library) assembles library of feature elements.

Classes: tei.metadata

Declaration:

```
element fLib { tei.global.attributes, fLib.attributes.type, f+ }
```

Attributes: (In addition to global attributes and those inherited from tei.metadata)

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

Example:

```
<fLib type="agreement features">
 <f id="p1" name="person">
  <sym value="first"/>
 </f>
 <f id="p2" name="person">
  <sym value="second"/>
 </f><!-- ... --><f id="ns" name="number">
```

```
  <sym value="singular"/>
 </f>
 <f id="np" name="number">
  <sym value="plural"/>
 </f><!-- ... -->
</fLib>
```

Module: iso-fs

4.6 fs [element]

Description: (Feature structure) represents a feature structure, that is, a collection of feature-value pairs organized as a structural unit.

Classes: tei.complexVal tei.metadata

Declaration:

```
element fs
{
   tei.global.attributes,
   fs.attributes.type,
   fs.attributes.feats,
   f*
}
```

Attributes: (In addition to global attributes and those inherited from tei.complexVal, tei.metadata)

type specifies the type of the feature structure.

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

Example:

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

Module: iso-fs

4.7 fvLib [element]

Description: (Feature-value library) assembles a library of reusable feature value elements (including complete feature structures).

Classes: tei.metadata

Declaration:

```
element fvLib { tei.global.attributes, fvLib.attributes.type, tei.featureVal* }
```

Attributes: (In addition to global attributes and those inherited from tei.metadata)

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

Example:

```
<fvLib type="symbolic values">
 <symbol id="sfirst" value="first"/>
 <symbol id="ssecond" value="second"/><!-- ... --><symbol id="ssing" value="singular"/>
 <symbol id="splur" value="plural"/><!-- ... -->
</fvLib>
```

A feature value library may include any number of values of any kind, including multiple occurrences of identical values such as `<binary value="true"/>` or `default`. The only thing guaranteed unique in a feature value library is the set of labels used to identify the values.

Module: iso-fs

4.8 numeric [element]

Description: (Numeric value) represents a numeric value or range of values for a feature.

Classes: tei.singleVal

Declaration:

```
element numeric
{
   tei.global.attributes,
   numeric.attributes.value,
   numeric.attributes.max,
   numeric.attributes.trunc,
   empty
}
```

Attributes: (In addition to global attributes and those inherited from tei.singleVal)

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.

Example:

```
<numeric value="42"/>
```

This represents the numeric value 42.

Example:

```
<numeric value="42.45" max="50" trunc="yes"/>
```

This represents any of the nine possible integer values between 42 and 50 inclusive. If the trunc attribute had the value NO, this example would represent any of the infinite number of numeric values between 42.45 and 50.0

It is an error to supply the max attribute in the absence of a value for the value attribute.

Module: iso-fs

4.9 string [element]

Description: (String value) provides a string value for a feature.

Classes: tei.singleVal

Declaration:

```
element string { tei.global.attributes, text }
```

Attributes: Global attributes and those inherited from tei.singleVal

Example:

```
<stringVal>Hello, world!</stringVal>
```

Module: iso-fs

4.10 symbol [element]

Description: (Symbolic value) provides a symbolic feature value.

Classes: tei.singleVal

Declaration:

```
element symbol { tei.global.attributes, symbol.attributes.value, empty }
```

Attributes: (In addition to global attributes and those inherited from tei.singleVal)

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

Example:

```
<symbol value="feminine"/>
```

Module: iso-fs

4.11 vAlt [element]

Description: (Value alternation) represents a feature value which is the disjunction of the values it contains.

Classes: tei.complexVal

Declaration:

```
element vAlt { tei.global.attributes, ( ( tei.featureVal ), tei.featureVal+ ) }
```

Attributes: Global attributes and those inherited from tei.complexVal

Example:

```
<vAlt>
 <sym value="masculine"/>
 <sym value="neuter"/>
</vAlt>
```

Module: iso-fs

4.12 vColl [element]

Description: (Collection of values) represents a feature value which is the result of collecting together the feature values it combines, using the organization specified by the ORG attribute.

Classes: tei.complexVal

Declaration:

```
element vColl { tei.global.attributes, vColl.attributes.org, tei.featureVal+ }
```

Attributes: (In addition to global attributes and those inherited from tei.complexVal)

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.

Example:

```
<vColl org="list">
 <coll org="set">
  <sym value="masculine"/>
  <sym value="neuter"/>
  <sym value="feminine"/>
 </coll>
 <sym value="indeterminate"/>
</vColl>
```

This example returns the concatenation of the indeterminate value with the set of values masculine, neuter and feminine.

Module: iso-fs

4.13 vNot [element]

Description: (Value negation) represents a feature value which is the negation of its content.

Classes: tei.complexVal

Declaration:

```
element vNot { tei.global.attributes, ( tei.featureVal ) }
```

Attributes: Global attributes and those inherited from tei.complexVal

Example:

```
<vNot>
 <sym value="masculine"/>
</vNot>
```

Example:

```
<f name="mode">
 <vNot>
  <vAlt>
   <symbol value="infinitive"/>
   <symbol value="participle"/>
  </vAlt>
 </vNot>
</f>
```

Module: iso-fs

4.14 var [element]

Description: (variable) indicates that the feature value is shared with other values bearing the same label within the current scope.

Classes: tei.singleVal

Declaration:

```
element var { tei.global.attributes, var.attributes.label, tei.featureVal? }
```

Attributes: (In addition to global attributes and those inherited from tei.singleVal)

label supplies a label for the variable.

Example:

```
<f name="n">
 <fs>
  <f name="index">
   <var label="L001"/>
  </f>
  <f name="restr">
   <var label="L002"/>
  </f>
 </fs>
</f>
```

Module: iso-fs