26 Feature System Declaration

The Feature System Declaration (FSD) is an auxiliary file used in conjunction with a TEI-conforming text that makes use of <fs> (that is, feature structure) elements. The FSD serves three purposes:

- It provides a mechanism by which the encoder can list all of the feature names and feature values and give a prose description as to what each represents.
- It provides a mechanism by which the encoder can define constraints on what it means to be a well-formed feature structure. These constraints may involve constraints on the range of a feature value, constraints on what features are valid within certain types of feature structures, or constraints that prevent the co-occurrence of certain feature-value pairs.
- It provides a mechanism by which the encoder can define the intended interpretation of underspecified feature structures. This involves defining default values (whether literal or computed) for missing features.

As a component of the interchange standard for encoded text, the FSD serves an important function in documenting precisely what the encoder intended by the system of feature structure markup used in the encoded text. As application software is developed which makes use of encoded texts, the FSD will also become an important resource that will allow software to validate the feature structure markup in a text and to infer the full interpretation of underspecified feature structures.

This chapter begins by describing how the encoded text uses header information to make links to any associated FSDs. The second through fourth sections describe the overall structure of an FSD and give details of how to encode its parts. The final section offers a full example.[166]

26.1 Linking a TEI Text to Feature System Declarations

In order for application software to use feature system declarations to aid in the automatic interpretation of encoded texts, or even for human readers to find the appropriate declarations which document the feature system used in markup, there must be a formal link from the encoded texts to the declarations. However, the auxiliary DTD which declares the syntax for the Feature System itself must also be kept distinct from the feature structure DTD, which is an application of that system.

In the present version of these Guidelines, this is accomplished by regarding the Feature System document (FSD) as an external unparsed entity, which is referenced only by name within the document that uses it.[167]

The association between an FSD and a document using the feature structures it declares is made in the following way. Firstly, an external unparsed entity must be declared for each FSD that is associated with the encoded text. That entity declaration gives a name for the external entity and associates it with some file or other resource on the host system. It must also contain the keyword NDATA, to indicate that the external entity contains unparsed XML or SGML data conforming to some other notation, and a previously-declared name for that notation. (In an SGML document, the SUBDOC keyword may be used in place of NDATA and the Notation name to tell the processor that the named file is a self-contained SGML document.) See the example below for details of syntax.

Secondly, the name of the relevant FSD entity will be referenced within the TEI header of a document containing feature structure annotation, as mentioned in section 5.3.7 *The Feature System Declaration*. Within the <encodingDesc> element of such a document's <teiHeader>, a special <fsdDecl> element may be used for each distinct feature structure type, as follows:

<fsdDecl> identifies the feature system declaration which contains definitions for a particular type of feature structure. Attributes include:

[166] For a fuller discussion of the reasoning behind FSDs and for another complete example, see *A rationale for the TEI recommendations for feature-structure markup*, by D. Terence Langendoen and Gary F. Simons , in *Computers and the Humanities*, 29, (1995).

[167] In SGML (but not in XML) a feature known as SUBDOC is available which allows a document using one DTD (the FSD) to be nested within another (the feature structure itself); this feature is not available in XML, and is therefore not recommended where usage of XML is intended.

type identifies the type of feature structure documented in the FSD; this will be the value of the type attribute on at least one feature structure.
Values any string of characters.

fsd (feature-system declaration) specifies the external entity containing the feature system declaration; an entity declaration in the document's DTD subset must associate the entity name with a file on the system.
Values a valid external entity name

Note that one <fsdDecl> element must be specified for each distinct type of feature structure used in the markup. The fsd attribute supplies the name of the external entity containing the actual declaration for that type of feature structure.

There may be multiple <fsdDecl> elements for a given FSD; one for each type of feature structure it defines. For instance, in the following example, the file lex.fsd contains an FSD that contains definitions of feature structures for both lexical entries (<fs type="entry">) and lexical subentries (<fs type="subentry">).

The following example shows the markup for linking an XML document to two FSDs.

```
<!DOCTYPE TEI.2 PUBLIC "-//TEI P4//DTD Main Document Type//EN"
                       "tei2.dtd" [
     <!-- Use prose base, with feature structure topping-->
     <!ENTITY % TEI.prose    'INCLUDE'>
     <!ENTITY % TEI.fs    'INCLUDE'>
     <!-- Use xml -->
     <!ENTITY % TEI.XML      'INCLUDE' >
     <!-- Declare the fsd notation itself -->
     <!NOTATION fsd
         PUBLIC "-//TEI//Feature System Declaration (1994)//EN">
     <!-- Now, declare external entities for our FSDs  -->
     <!ENTITY fsdGazdar  SYSTEM 'gpsg.fsd' NDATA fsd >
     <!ENTITY fsdLexicon SYSTEM 'lex.fsd'  NDATA fsd >
  ]>
  <TEI.2>

  <teiHeader>
     <fileDesc> ... </fileDesc>
     <encodingDesc>
          <!-- ... -->
          <fsdDecl type='GPSG'     fsd='fsdGazdar'>
          <fsdDecl type='entry'    fsd='fsdLexicon'>
          <fsdDecl type='subentry' fsd='fsdLexicon'>
          <!-- ... -->
     </encodingDesc>
  </teiHeader>
  <!-- The text goes here -->
  </TEI.2>
```

As this example shows, a <fsdDecl> is given within the <encodingDesc> for each distinct value used as the type of the <fs> elements in the document itself. In this case, for example, the feature system declaration used by feature structures of types entry and subentry is to be found in the entity named fsdLexicon, previously associated with the system file lex.fsd.

The original version of the TEI Guidelines did not enforce uniqueness of the type values for the <fsdDecl> element, nor did they require that every type value specified on a <fs> element also be declared on an <fsdDecl> element. These constraints have some obvious utility in assisting the consistency and accuracy of tagging; however to enforce them with the current DTD would require changing the declared value for the current<fs> element from CDATA to IDREF, as well as restricting the possible values for type to legal identifiers rather than meaningful strings. The encoder wishing to apply such constraints with the current DTDs is recommended to do so by using the existing id attribute in place of the <type> attribute on the <fsdDecl> element, and declaring a new fsd attribute of type IDREF on the <fs> element. These changes, or ones designed to achieve similar effect, may be made in a subsequent version of these Guidelines.

The auxiliary tag set for feature system declarations is contained in the file teifsd2.dtd, which has the public identifier -//TEI P4//DTD Auxiliary Document Type: Feature System Declaration//EN and the overall structure shown below:

```
<!-- 26.1: Feature System Declaration-->
<!--Text Encoding Initiative Consortium:
Guidelines for Electronic Text Encoding and Interchange.
Document TEI P4, 2002.
Copyright (c) 2002 TEI Consortium. Permission to copy in any form
is granted, provided this notice is included in all copies.
These materials may not be altered; modifications to these DTDs should
be performed only as specified by the Guidelines, for example in the
chapter entitled 'Modifying the TEI DTD'
These materials are subject to revision by the TEI Consortium. Current versions
are available from the Consortium website at http://www.tei-c.org-->
<!--First, we declare basic parameter entities and entities for
TEI generic identifiers.-->
<!ENTITY % TEI.elementNames PUBLIC '-//TEI P4//ENTITIES Generic
Identifiers//EN' 'teigis2.ent' >%TEI.elementNames;
<!--Declare entities for TEI keywords.-->
<!ENTITY % TEI.keywords.ent PUBLIC '-//TEI P4//ENTITIES TEI
Keywords//EN' 'teikey2.ent' >%TEI.keywords.ent;
<!--Declare element classes for content models, shared
attributes for element classes, and global attributes.  (This all
happens within the file teiclas2.ent.)-->
<!ENTITY % TEI.elementClasses PUBLIC '-//TEI P4//ENTITIES TEI
ElementClasses//EN' 'teiclas2.ent' >%TEI.elementClasses;
<!--Declare element classes for feature structure
declarations.-->
<!ENTITY % x.boolean "" >
<!ENTITY % m.boolean "%x.boolean; %n.any; | %n.none;">
<!ENTITY % x.binary "" >
<!ENTITY % m.binary "%x.binary; %n.minus; | %n.plus;">
<!ENTITY % x.singleVal "" >
<!ENTITY % m.singleVal "%x.singleVal; %m.binary; | %m.boolean; | %n.dft; |
%n.msr; | %n.nbr; | %n.rate; | %n.str; | %n.sym; | %n.uncertain;">
<!ENTITY % x.complexVal "" >
<!ENTITY % m.complexVal "%x.complexVal; %n.alt; | %n.fs; | %n.vAlt;">
<!ENTITY % x.featureVal "" >
<!ENTITY % m.featureVal "%x.featureVal; %m.complexVal; | %n.null; |
%m.singleVal;">
<!--Now, we declare the elements for FSDs proper.-->
<!--declarations from 26.2: Feature System Declaration inserted here -->
<!--declarations from 26.3: Feature definitions inserted here -->
<!--declarations from 26.4: Feature structure constraints inserted here -->
<!--The elements for feature structures themselves
are declared in teifs2.dtd-->
<!ENTITY % TEI.fs.dtd PUBLIC '-//TEI P4//DTD Auxiliary Document Type:
 Feature System Declaration//EN' 'teifs2.dtd' >%TEI.fs.dtd;
<!--Finally, declare the TEI header and core tag
sets.-->
<!ENTITY % TEI.header.dtd PUBLIC '-//TEI P4//ELEMENTS TEI Header//EN'
'teihdr2.dtd' >%TEI.header.dtd;

<!ENTITY % TEI.core.dtd PUBLIC '-//TEI P4//ELEMENTS Core Elements//EN'
'teicore2.dtd' >%TEI.core.dtd;
<!-- end of 26.1-->
```

26.2 The Overall Structure of a Feature System Declaration

A feature system declaration is encoded as a document of type <teiFsd2>. It has two parts: an obligatory header (which provides bibliographic information for the file) and a set of feature structure declarations (each of which defines one type of feature structure). Each feature structure declaration in turn has three parts: an optional description (which gives a prose comment on what that type of feature structure encodes), an obligatory set of feature declarations (which specify range constraints and default values

for the features in that type of structure), and optional feature structure constraints (which specify co-occurrence restrictions on feature values). The header is encoded as a <teiHeader>, just as for any TEI.2 document; see chapter 5 *The TEI Header*. The other components listed above are unique to feature system declarations. Thus, the following new elements are involved:

<teiFsd2> contains a feature system declaration.

<fsDecl> declares one type of feature structure. Attributes include:

type gives a name for the type of feature structure being declared.
Values any convenient string of characters.

baseType gives the name of the feature structure type from which this type inherits features and constraints; if this type declares a feature with the same name as a feature of the base type, the definition within this <fsDecl> overrides the inherited definition. The <fsConstraints> are inherited only if this <fsDecl> does not specify any; otherwise the constraints in this <fsDecl> override. When no baseType is specified, no features or constraints are inherited.
Values any convenient string for use as a name.

<fsDescr> describes in prose what is represented by the type of feature structure declared in the enclosing <fsDecl>.

<fDecl> declares a single feature, specifying its name, organization, range of allowed values, and optionally its default value. Attributes include:

name indicates the name of the feature being declared; matches the name attribute of <f> elements in the text.
Values any string of characters

org (organization) specifies the organizing discipline of the feature value.
Legal values are:

- unit unitary atomic value
- set set value (unordered, no duplicates)
- bag bag value (unordered, may have duplicates)
- list list value (ordered, may have duplicates)

<fsConstraints> specifies constraints on the content of well formed feature structures.

Feature declarations and feature structure constraints are described in the next two sections of this chapter. Note that the specification of similar <fsDecl> elements can be simplified by devising an inheritance hierarchy for the feature structure types. Each <fsDecl> may name a baseType from which it inherits feature declarations and constraints. For instance, suppose that <fsDecl type="Basic"> contains <fDecl name="One"> and <fDecl name="Two">, and that <fsDecl type="Derived" baseType="Basic"> contains just <fDecl name="Three">. Then any instance of <fs type="Derived"> may include all three features. This is because <fsDecl type="Derived"> inherits the two feature declarations from <fsDecl type="Basic"> when it specifies a baseType of Basic.

The following sample shows the overall structure of a complete FSD. Note that as a stand-alone document it begins with a DOCTYPE declaration which identifies the associated DTD.

```
<!DOCTYPE teiFsd2 PUBLIC "-//TEI P4//DTD Auxiliary Document Type:
        Feature System Declaration//EN"
       "teifsd2.dtd" [
  <!ENTITY % TEI.XML      'INCLUDE' >
]>
  <teiFsd2>
     <teiHeader>
        <!-- The header is as for any TEI.2 document -->
     </teiHeader>
     <fsDecl type='SomeName'>
        <fsDescr>Describes what this type of fs represents</fsDescr>
        <fDecl name='featureOne'>
           <!-- The declaration for featureOne -->
        </fDecl>
        <fDecl name='featureTwo'>
           <!-- The declaration for featureTwo -->
```

```
        </fDecl>
        <fsConstraints>
           <!-- The feature structure constraints go here -->
        </fsConstraints>
      </fsDecl>
      <fsDecl type='AnotherType'>
            <!-- Declare another type of feature structure -->
      </fsDecl>
    </teiFsd2>
```

The formal definition of <teiFsd2> and feature structure declarations is as follows:

```
<!-- 26.2: Feature System Declaration-->
<!ELEMENT teiFsd2 %om.RR; (teiHeader, fsDecl+)>
<!ATTLIST teiFsd2
      %a.global;
      TEIform CDATA 'teiFsd2'  >
<!ELEMENT fsDecl %om.RR;  (fsDescr?, fDecl+, fsConstraints?)>
<!ATTLIST fsDecl
      %a.global;
      type CDATA #REQUIRED
      baseType CDATA #IMPLIED
      TEIform CDATA 'fsDecl'  >
<!ELEMENT fsDescr %om.RO;  %paraContent;>
<!ATTLIST fsDescr
      %a.global;
      TEIform CDATA 'fsDescr'  >
<!-- end of 26.2-->
```

26.3 Feature Declarations

Each feature is declared in an <fDecl> element whose name attribute identifies the feature being declared; this matches the name attribute of the <f> elements it declares. An <fDecl> also has an org attribute which declares the organizing principle for the values of the <f> elements it declares. That is, the value may be a unit (a single value), a set (in which the order is not significant and there are no duplicates), a bag (in which the order is not significant but duplicates are allowed), or a list (in which the order is significant). (See definition of org attribute of <f> in section 16.6 *Singleton, Set, Bag and List Collections of Values*.) An <fDecl> has three parts: an optional prose description (which should explain what the feature and its values represent), an obligatory range specification (which declares what values the feature is allowed to have), and an optional default specification (which declares what default value should be supplied when the named feature does not appear in an <fs>). A single unconditional default value may be specified, or multiple conditional values. If no default is specified, or if none of the conditions is met, then the default value is <none>; in other words, the feature is not applicable (see section 16.8 *Boolean, Default and Uncertain Values* for a discussion of the <none> element).

The tags used in feature declarations are the following:

<fDecl> declares a single feature, specifying its name, organization, range of allowed values, and optionally its default value. Attributes include:

name indicates the name of the feature being declared; matches the name attribute of <f> elements in the text.

Values any string of characters

org (organization) specifies the organizing discipline of the feature value.

Legal values are:

- unit unitary atomic value
- set set value (unordered, no duplicates)
- bag bag value (unordered, may have duplicates)
- list list value (ordered, may have duplicates)

<fDescr> describes in prose what is represented by the feature being declared and its values.

<vRange> defines the range of allowed values for a feature, in the form of an <fs>, <vAlt>, or primitive value; for the value of an <f> to be valid, it must be *subsumed* by the specified range; if the <f> contains multiple values (as sanctioned by the org attribute), then each value must be subsumed by the <vRange>.

<vDefault> declares the default value to be supplied when a feature structure does not contain an instance of <f> for this name; if unconditional, it is specified as one (or, depending on the value of the org attribute of the enclosing <fDecl>) more <fs> elements or primitive values; if conditional, it is specified as one or more <if> elements; if no default is specified, or no condition matches, the value <none> is assumed.

<if> defines a conditional default value for a feature; the condition is specified as a feature structure, and is met if it *subsumes* the feature structure in the text for which a default value is sought.

<then> separates the condition from the default in an <if>, or the antecedent and the consequent in a <cond> element.

The logic for validating feature values and for matching the conditions for supplying default values is based on the operation of subsumption. Subsumption is a standard operation in feature-structure-based formalisms. Informally, a feature structure *fs* subsumes all feature structures that are at least as informative as itself; that is, all feature structures that specify at least as many features as *fs* with values at least as informative as those given in *fs* (Pereira 1987:6; see also Shieber 1986:14–16).[168] A more formal definition requires that we first define the notion of "domain of a feature structure." A feature structure can be viewed as a partial function that maps features onto values; when viewed in this way, the domain of a feature structure is the set of top-level features it contains (that is, excluding features in embedded feature structures). We can now offer a more precise definition:

> *fs* subsumes *fs′* if both are identical primitive values, or if the domain of *fs* is a subset of the domain of *fs′*, and for every feature *f* in the domain of *fs*, the value of *f* in *fs* subsumes the value of *f* in *fs′*.

Following the spirit of the informal definition above, we can extend subsumption in a straightforward way to cover alternation, negation, special primitive values, and the use of attributes in the markup. For instance, a <vAlt> containing the value v subsumes v. The negation REL='ne' of value v subsumes any value that is not v. The value <unknown> subsumes any value. The value <any> subsumes any value that is in the range of a feature. <fs type="X"/> subsumes any feature structure of type X. <nbr rel='ge' value='0'/> subsumes any <nbr> element with value greater than or equal to zero.

As an example of feature declarations, consider the following extract from Gazdar et al's *Generalized Phrase Structure Grammar*.[169] In the appendix to their book (pages 245–247), they propose a feature system for English of which this is just a sampling:

```
feature    value range
INV        {+, -}
CONJ       {and, both, but, either, neither, nor, or, NIL}
COMP       {for, that, whether, if, NIL}
AGR        CAT
PFORM      {to, by, for, ...}

Feature specification defaults
FSD 1:  [-INV]
FSD 2:  ~[CONJ]
FSD 9:  [INF, +SUBJ] --> [COMP for]
```

The INV feature, which encodes whether or not a sentence is inverted, allows only the values plus (+) and minus (-). If the feature is not specified, then the default rule (FSD 1 above) says that a value of minus is always assumed. The feature declaration for this feature would be encoded as follows:

```
<fDecl name='INV'>
  <fDescr>inverted sentence</fDescr>
   <vRange>
```

[168] Fernando C. N. Pereira, *Grammars and logics of partial information*, SRI International Technical Note 420 (Menlo Park, CA: SRI International, 1987), and Stuart Shieber, *An Introduction to Unification-based Approaches to Grammar*, CSLI Lecture Notes 4 (Palo Alto, CA: Center for the Study of Language and Information, 1986).

[169] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag: *Generalized Phrase Structure Grammar*, (Harvard University Press, 1985)

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

The value range is specified as an alternation (more precisely, an exclusive disjunction) of <plus/> and <minus/>. That is, the value must be one or the other, but not both or neither.

The CONJ feature indicates the surface form of the conjunction used in a construction. The ~ in the default rule (see FSD 2 above) represents negation. This means that by default the feature is not applicable, in other words, no conjunction is taking place. This corresponds to the simple value <none>; see section 16.8 *Boolean, Default and Uncertain Values*. Note that this is distinct from the NIL value allowed in the value range. In their analysis, NIL means that the phenomenon of conjunction is taking place but there is no explicit conjunction in the surface form of the sentence. The feature declaration for this feature would be encoded as follows:

```
<fDecl name='CONJ'>
   <fDescr>surface form of the conjunction</fDescr>
   <vRange>
      <vAlt>
        <sym value='and'/>
        <sym value='both'/>
        <sym value='but'/>
        <sym value='either'/>
        <sym value='neither'/>
        <sym value='nor'/>
        <sym value='or'/>
        <sym value='NIL'/>
      </vAlt>
   </vRange>
   <vDefault><none/></vDefault>
</fDecl>
```

Note that the <vDefault> is not strictly necessary in this case, since <none> is the value assumed in the absence of a default specification.

The COMP feature indicates the surface form of the complementizer used in a construction. In value range, it is analogous to CONJ. However, its default rule (see FSD 9 above) is conditional. It says that if the verb form is infinitival (the VFORM feature is not mentioned in the rule since it is the only feature that can take INF as a value), and the construction has a subject, then a 'for' complement must be used. For instance, to make John the subject of the infinitive in 'It is necessary to go,' a 'for' complement must be used; that is, 'It is necessary for John to go.' The feature declaration for this feature would be encoded as follows:

```
<fDecl name='COMP'>
   <fDescr>surface form of the complementizer</fDescr>
   <vRange>
      <vAlt>
        <sym value='for'/>
        <sym value='that'/>
        <sym value='whether'/>
        <sym value='if'/>
        <sym value='NIL'/>
      </vAlt></vRange>
   <vDefault>
      <if><fs><f name='VFORM'><sym value='INF'/></f>
              <f name='SUBJ'><plus/></f></fs>
      <then/><sym value='for'/></if>
   </vDefault>
</fDecl>
```

The AGR feature stores the features relevant to subject-verb agreement. Gazdar et al. specify the range of this feature as CAT. This means that the value is a *category*, which is their term for a feature structure. This is actually too weak a statement. Not just any feature structure is allowable here; it must be a feature structure for agreement (which is defined in the complete example at the end of the chapter to contain the

features of person and number). The following feature declaration encodes this constraint on the value range:

```
<fDecl name='AGR'>
   <fDescr>agreement for person and number</fDescr>
   <vRange><fs type='Agreement'></fs></vRange>
</fDecl>
```

That is, the value must be a feature structure of type `Agreement`. The complete example at the end of this chapter includes the `<fsDecl type="Agreement">` which includes `<fDecl name="PERS">` and `<fDecl name="NUM">`.

The PFORM feature indicates the surface form of the preposition used in a construction. Since PFORM is specified above as an open set, `<str>` is used in the range specification below rather than `<sym>`.

```
<fDecl name='PFORM'>
   <fDescr>word form of a preposition</fDescr>
   <vRange><str rel='ne'></str></vRange>
</fDecl>
```

This example makes use of a negation. `<str rel="ne"></str>` subsumes any string that is not the empty string.

The formal definition for feature declarations follows. Note that the class `featureVal` includes all possible single feature values, including a `<vAlt>`.

```
<!-- 26.3: Feature definitions-->
<!ELEMENT fDecl %om.RR;  (fDescr?, vRange, vDefault?)>
<!ATTLIST fDecl
      %a.global;
      name NMTOKEN #REQUIRED
      org (unit | set | bag | list) "unit"
      TEIform CDATA 'fDecl'  >
<!ELEMENT fDescr %om.RO;  %paraContent;>
<!ATTLIST fDescr
      %a.global;
      TEIform CDATA 'fDescr'  >
<!ELEMENT vRange %om.RO;  (%m.featureVal;)>
<!ATTLIST vRange
      %a.global;
      TEIform CDATA 'vRange'  >
<!ELEMENT vDefault %om.RR; ((%m.featureVal;)+ | if+)>
<!ATTLIST vDefault
      %a.global;
      TEIform CDATA 'vDefault'  >
<!ELEMENT if %om.RR;  ((fs | f | fAlt), then, (%m.featureVal;) )>
<!ATTLIST if
      %a.global;
      TEIform CDATA 'if'  >
<!ELEMENT then %om.RO;  EMPTY>
<!ATTLIST then
      %a.global;
      TEIform CDATA 'then'  >
<!-- end of 26.3-->
```

26.4 Feature Structure Constraints

Ensuring the validity of feature structures may require much more than simply specifying the range of allowed values for each feature. There may be constraints on the co-occurrence of one feature value with the value of another feature in the same feature structure or in an embedded feature structure.

Such constraints on valid feature structures are expressed as a series of conditional and biconditional tests in the `<fsConstraints>` part of an `<fsDecl>`. A particular feature structure is valid only if it meets all the constraints. The `<cond>` element encodes the conventional if-then conditional of boolean logic which succeeds when both the antecedent and consequent are true, or whenever the antecedent is false. The `<bicond>` element encodes the biconditional (if and only if) operation of boolean logic. It succeeds

only when both antecedent and consequent are true, or both are false. In feature structure constraints the antecedent and consequent are expressed as feature structures; they are considered true if they *subsume* (see section 26.3 *Feature Declarations*) the target feature structure. The following elements make up the <fsConstraints> part of an FSD:

<fsConstraints> specifies constraints on the content of well formed feature structures.

<cond> defines a conditional feature-structure constraint; the consequent and the antecedent are specified as feature structures or feature-structure groups; the constraint is satisfied if both the antecedent and the consequent *subsume* a given feature structure, or if the antecedent does not.

<bicond> defines a biconditional feature-structure constraint; both consequent and antecedent are specified as feature structures or groups of feature structures; the constraint is satisfied if both *subsume* a given feature structure, or if both do not.

<then> separates the condition from the default in an <if>, or the antecedent and the consequent in a <cond> element.

<iff> separates the condition from the consequence in a <bicond> element.

For an example of feature structure constraints, consider the following 'feature co-occurrence restrictions' extracted from the feature system for English proposed by Gazdar, Klein, Pullum, and Sag (1985:246–247):

```
FCR 1:  [+INV] &#8594; [+AUX, FIN]
FCR 7:  [BAR 0] &equiv; [N] &amp; [V] &amp; [SUBCAT]
FCR 8:  [BAR 1] &#8594; ~[SUBCAT]
```

The first constraint says that if a construction is inverted, it must also have an auxiliary and a finite verb form. That is,

```
<cond>
   <fs><f name='INV'><plus/></f></fs>
 <then/>
   <fs><f name='AUX'><plus/></f>
       <f name='VFORM'><sym value='FIN'/></f>
   </fs>
</cond>
```

The second constraint says that if a construction has a BAR value of zero (i.e., it is a sentence), then it must have a value for the features N, V, and SUBCAT. By the same token, because it is a biconditional, if it has values for N, V, and SUBCAT, it must have BAR='0'. That is,

```
<bicond>
   <fs><f name='BAR'><sym value='0'/></f></fs>
   <iff/>
   <fs>
      <f name='N'><any/></f>
      <f name='V'><any/></f>
      <f name='SUBCAT'><any/></f>
   </fs>
</bicond>
```

The final constraint says that if a construction has a BAR value of 1 (i.e., it is a phrase), then the SUBCAT feature is irrelevant (~). This is not biconditional, since there are other instances under which the SUBCAT feature is irrelevant. That is,

```
<cond>
   <fs><f name='BAR'><sym value='1'/></f></fs>
  <then/>
    <fs><f name='SUBCAT'><none/></f></fs>
</cond>
```

The DTD fragment for feature structure constraints is as follows. Note that <cond> and <bicond> use the empty tags <then> and <iff>, respectively, to separate the antecedent and consequent. These are primarily for the sake of enhancing human readability.

```
<!-- 26.4: Feature structure constraints-->
<!ELEMENT fsConstraints %om.RR;  (cond | bicond)*>
<!ATTLIST fsConstraints
```

```
      %a.global;
      TEIform CDATA 'fsConstraints'  >
<!ELEMENT cond %om.RO;  ((fs | f | fAlt), then, (fs | f | fAlt))>
<!ATTLIST cond
      %a.global;
      TEIform CDATA 'cond'  >
<!ELEMENT bicond %om.RO;  ((fs | f | fAlt), iff, (fs | f | fAlt))>
<!ATTLIST bicond
      %a.global;
      TEIform CDATA 'bicond'  >
<!ELEMENT iff %om.RO;  EMPTY>
<!ATTLIST iff
      %a.global;
      TEIform CDATA 'iff'  >
<!-- end of 26.4-->
```

26.5 A Complete Example

To summarize this chapter, the complete FSD for the example that has run through the chapter is reproduced below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE teiFsd2 SYSTEM "teifsd2.dtd" [
  <!ENTITY % TEI.XML  'INCLUDE' >
]>
<teiFsd2>
<teiHeader>
<fileDesc>
<titleStmt>
   <title>A sample FSD based on an extract from Gazdar
      et al.'s GPSG feature system for English</title>
   <respStmt>
      <resp>encoded by</resp>
      <name>Gary F. Simons</name>
   </respStmt>
</titleStmt>
<publicationStmt>
<p>This sample was first encoded by Gary F. Simons (Summer
Institute of Linguistics, Dallas, TX) on January 28, 1991.
Revised April 8, 1993 to match the specification of FSDs
in version P2 of the TEI Guidelines.
</p></publicationStmt>
<sourceDesc>
<p>This sample FSD does not describe a complete feature
system.  It is based on extracts from the feature system
for English presented in the appendix (pages 245&#x2013;247) of
Generalized Phrase Structure Grammar, by Gazdar, Klein,
Pullum, and Sag (Harvard University Press, 1985).</p>
</sourceDesc>
</fileDesc>
</teiHeader>

<fsDecl type='GPSG'>
   <fsDescr>Encodes a feature structure for the GPSG analysis
     of English (after Gazdar, Klein, Pullum, and Sag)</fsDescr>
   <fDecl name='INV'>
      <fDescr>inverted sentence</fDescr>
      <vRange>
         <vAlt><plus/><minus/></vAlt></vRange>
      <vDefault><minus/></vDefault>
   </fDecl>
   <fDecl name='CONJ'>
      <fDescr>surface form of the conjunction</fDescr>
      <vRange><vAlt><sym value='and'/><sym value='both'/>
         <sym value='but'/><sym value='either'/>
         <sym value='neither'/><sym value='nor'/>
```

```
          <sym value='or'/><sym value='NIL'/>
        </vAlt></vRange>
        <vDefault><none/></vDefault>
        </fDecl>
     <fDecl name='COMP'>
        <fDescr>surface form of the complementizer</fDescr>
        <vRange><vAlt><sym value='for'/>
             <sym value='that'/><sym value='whether'/>
             <sym value='if'/><sym value='NIL'/>
         </vAlt></vRange>
        <vDefault>
           <if><fs><f name='VFORM'><sym value='INF'/></f>
                   <f name='SUBJ'><plus/></f></fs>
          <then/><sym value='for'/></if>
        </vDefault>
     </fDecl>
     <fDecl name='AGR'>
        <fDescr>agreement for person and number</fDescr>
        <vRange><fs type='Agreement'></fs></vRange>
     </fDecl>
     <fDecl name='PFORM'>
        <fDescr>word form of a preposition</fDescr>
        <vRange><str rel='ne'></str></vRange>
     </fDecl>

     <fsConstraints>
        <cond><fs><f name='INV'><plus/></f></fs>
        <then/><fs>
            <f name='AUX'><plus/></f>
            <f name='VFORM'><sym value='FIN'/></f>
            </fs>
       </cond>
       <bicond><fs><f name='BAR'><sym value='0'/></f></fs>
        <iff/><fs>
           <f name='N'><any/></f>
           <f name='V'><any/></f>
           <f name='SUBCAT'><any/></f>
            </fs>
        </bicond>
        <cond><fs><f name='BAR'><sym value='1'/></f></fs>
          <then/>
             <fs><f name='SUBCAT'><none/></f></fs>
        </cond>
     </fsConstraints>
  </fsDecl>

  <fsDecl type='Agreement'>
     <fsDescr>This type of feature structure encodes the features
        for subject-verb agreement in English</fsDescr>
     <fDecl name='PERS'>
        <fDescr>person (first, second, or third)</fDescr>
        <vRange><vAlt>
          <sym value='1'/><sym value='2'/><sym value='3'/>
         </vAlt></vRange>
     </fDecl>
     <fDecl name='NUM'>
        <fDescr>number (singular or plural)</fDescr>
        <vRange><vAlt><sym value='sg'/><sym value='pl'/>
        </vAlt></vRange>
     </fDecl>
  </fsDecl>
  </teiFsd2>
```