The Association for Computers and the Humanities (ACH)

The Association for Computational Linguistics (ACL)

The Association for Literary and Linguistic Computing (ALLC)

Guidelines for Electronic Text Encoding and Interchange

Edited by

C. M. Sperberg-McQueen and Lou Burnard

TEI P2, Chapter 23 GD
Text Encoding Initiative
Chicago, Oxford

May 28, 1993

Draft Version 2, May 28, 1993

Note

This is the current draft (as of May 28, 1993) of one section of the *Guidelines for Electronic Text Encoding and Interchange*, version 2 (also known as TEI P2), produced by the Text Encoding Initiative. This fascicle contains:

- a table of contents for TEI P2, indicating the status of each section as of the release date of this fascicle
- a draft for one section of TEI P2
- entries derived from this section for the reference list of tags of TEI P2 (if any)
- a User Response and Comment Form

Please feel free to send us your comments on the contents of this draft and their applicability to your needs. Comments may be sent to the TEI-L list server for general discussion, or directly to either of the TEI editors (adresses on the User Response and Comment form). Bear in mind in reading, however, that TEI P2 is a reference manual, not a tutorial in the use of electronic text, and unavoidably assumes either a certain amount of prior knowledge or patience with unfamiliar technical material. Note also that some matters which may appear obscure in one section may well be treated in more detail elsewhere in the full text of TEI P2.

TEI P2, like P1, is a working draft. The observant will find several typographic and presentational flaws in the current text: for example, cross-references are not properly resolved in the LaTeX versions and some parts of the reference material are not yet complete. We hope to rectify as many as possible of these flaws in the coming months, but have decided not to allow them to delay any further the release of the draft. We hope that those interested in the Guidelines will find the current state of the text readable and usable enough for the purpose of substantive comment. As far as possible, all comments received on this section will be taken into account in the preparation of the next version of the Guidelines, which is due by the end of 1993.

Updates on the status of the draft as a whole will be distributed automatically to all subscribers of the Listserv list TEI-L. To subscribe to TEI-L, send electronic mail to the address LISTSERV@UICVM (or Listserv@uicvm.uic.edu) containing the single line

```
subscribe tei-l J. Smith
```

(substituting your name for “J. Smith”). TEI-L is also the appropriate place to pose questions or offer public comments on the TEI guidelines and other relevant issues. The TEI-L file server contains all the sections of TEI P2 thus far released, as well as other TEI materials. Its contents are shadowed at various other sites around the world; for more information consult one of the following documents:

- TEI ED J8, “Obtaining the Second Version of the TEI Guidelines (TEI P2)” (describes how to retrieve electronic copies of TEI P2 and the various formats they are available in)
- TEI ED J9, “Obtaining Paper Copies of the Second Version of the TEI Guidelines (TEI P2)” (describes how to request paper copies of TEI P2, for those without electronic mail access)

These documents are available from the TEI-L file server or through the editors at the addresses given on the User Response and Comment Form.

Overall Table of Contents for TEI P2

The following list shows the expected contents of TEI P2 as a whole and indicates the current status of each chapter.

Contents

Part 4

Additional Tag Sets

Chapter 23

Graphs, Networks, and Trees

Graphical representations are widely used for displaying relations among informational units because they help readers to visualize those relations and hence to understand them better. Two general types of graphical representations may be distinguished.[1]

- Graphs, in the strictly mathematical sense, consist of *points*, often called *nodes* or *vertices*, and connections among them, called *arcs*, or under certain conditions, *edges*. Among the various types of graphs are *networks* and *trees*. Graphs generally and networks in particular are dealt with directly below. Trees are dealt with separately in sections 23.2, "Trees," on page 15 and 23.3, "Another Tree Notation," on page 19.[2]
- *Charts*, which typically plot data in two or more dimensions, including plots with orthogonal or radial axes, bar charts, pie charts, and the like. These can be described using the elements defined in the additional tag set for figures and graphics; see chapter 24, "Tables, Equations, and Graphics," [in separate fascicle].

The following DTD fragment shows the overall organization of the tag set discussed in the remainder of this chapter.

```
<!-- 23:  Graphs, networks and trees                        -->
<!-- ... declarations from section 23.1                     -->
<!--     (Graphs)                                           -->
<!--     go here ...                                        -->
<!-- ... declarations from section 23.2                     -->
<!--     (Trees (basic method))                             -->
<!--     go here ...                                        -->
<!-- ... declarations from section 23.3                     -->
<!--     (Trees (alternate method))                         -->
<!--     go here ...                                        -->
```

This tag set is made available as described in 3.3, "Invocation of TEI DTDs," [in separate fascicle]; in a document which uses the markup described in this chapter, the document type declaration should contain the following declaration of the entity *TEI.nets*, or an equivalent one:

---

1 The primary responsibility for this chapter was borne by the working committee for analysis and interpretation, whose members are listed in the front matter. It was drafted by D. Terence Langendoen with assistance from Steven Zepp. The Sasine text example and suggested markup was provided by Daniel Greenstein.

2 The treatment here is largely based on the characterizations of graph types in Gary Chartrand and Linda Lesniak, *Graphs and Digraphs* (Menlo Park, CA: Wadsworth, 1986).

```
<!ENTITY % TEI.nets 'INCLUDE'>
```

The entire document type declaration for a document using this additional tag set might look like this:

```
<!DOCTYPE TEI.2 system 'tei2.dtd' [
   <!ENTITY % TEI.prose    'INCLUDE' >
   <!ENTITY % TEI.nets 'INCLUDE' >
]>
```

Among the types of qualitative relations often represented by graphs are organizational hierarchies, flow charts, genealogies, semantic networks, transition networks, grammatical relations, tournament schedules, seating plans, and directions to people's houses. In developing recommendations for the encoding of graphs of various types, we have relied on their formal, mathematical definitions and on the most common conventions for representing them visually. However, it must be emphasized that these recommendations do not provide for the full range of possible graphical representations, and deal only partially with questions of design, layout and placement.

23.1 Graphs and Digraphs

Broadly speaking, graphs can be divided into two types: *undirected* and *directed*. An undirected graph is a set of *nodes* (or *vertices*) together with a set of pairs of those vertices, called *arcs*, (or *edges*). Each node in an arc of an undirected graph is said to be *incident* with that arc, and the two vertices which make up an arc are said to be *adjacent*. An directed graph is like an undirected graph except that the arcs are *ordered pairs* of nodes. In the case of directed graphs, the term *edge* is not used; moreover, each arc in an directed graph is said to be *adjacent from* the node from which the arc emanates, and *adjacent to* the node to which the arc is directed. We use the element **<graph>** to encode graphs as a whole, **<node>** to encode nodes or vertices, and **<arc>** to encode arcs or edges; arcs can also be encoded by attributes on the **<node>** element. These elements have the following descriptions and attributes:

<graph> : encodes a graph, which is a collection of nodes, and arcs which connect the nodes. Attributes include:
- *type* : describes the type of graph. Sample values include:
  - *undirected* : undirected graph
  - *directed* : directed graph
  - *transition network* : a directed graph with distinguished initial and final nodes
  - *transducer* : a transition network with up to two labels on each arc
- *label* : gives a label for a graph.
- *order* : states the order of the graph, i.e., the number of its nodes.
- *size* : states the size of the graph, i.e., the number of its arcs.

<node> : encodes a node, a possibly labeled point in a graph. Attributes include:
- *label* : gives a label for a node.
- *label2* : gives a second label for a node.
- *value* : provides the value of a node, which is a feature structure or other analytic element.
- *type* : provides a type for a node. Sample values include:

initial : initial node in a transition network
final : final node in a transition network
adjFrom : gives the IDs of the nodes which are adjacent from the current node.
adjTo : gives the IDs of the nodes which are adjacent to the current node.
adj : gives the IDs of the nodes which are both adjacent to and adjacent from the current node.
inDegree : gives the in degree of the node, the number of nodes which are adjacent from the given node.
outDegree : gives the out degree of the node, the number of nodes which are adjacent to the given node.
degree : gives the degree of the node, the number of arcs with which the node is incident.

<arc> : encodes an arc, the connection from one node to another in a graph. Attributes include:
label : gives a label for an arc.
label2 : gives a second label for an arc.
from : gives the ID of the node which is adjacent from this arc.
to : gives the ID of the node which is adjacent to this arc.

Before proceeding, some additional terminology may be helpful. We define a *path* in a graph as a sequence of nodes n1, ..., nk such that there is an arc from each ni to ni+1 in the sequence. A *cyclic path*, or *cycle* is a path leading from a particular node back to itself. A graph that contains at least one cycle is said to be *cyclic*; otherwise it is *acyclic*. We say, finally, that a graph is *connected* if there is a path from some node to every other node in the graph; any graph that is not connected is said to be *disconnected.*

Here is an example of an undirected, cyclic disconnected graph, in which the nodes are labeled with three-letter codes for airports, and the arcs connecting the nodes are represented by horizontal and vertical lines, with 90 degree bends used simply to avoid having to draw diagonal lines.

```
 ,---LAX---,
 |         |
LVG-------PHX---TUS    CIB

Airline Connections in Southwestern USA
```

Next is a markup of the graph, using **<arc>** elements to encode the arcs.

```
<graph type=undirected
       id=cug1
       label='Airline Connections in Southwestern USA'
       rend='LABEL-PLACE bottom center NODE-FRAME none ARC solid line'
       order=5
       size=4>
  <node label=LAX id=LAX degree=2>
  <node label=LVG id=LVG degree=2>
  <node label=PHX id=PHX degree=3>
  <node label=TUS id=TUS degree=1>
  <node label=CIB id=CIB degree=0>
  <arc  from=LAX to=LVG>
```

```
  <arc  from=LAX to=PHX>
  <arc  from=LVG to=PHX>
  <arc  from=PHX to=TUS>
  </graph>
```

The *rend* attribute on the **<graph>** element is used here to encode certain information about how the graph is to be drawn. The *label* attribute on the **<graph>** element records the label of the graph (its placement being specified as part of the *rend* attribute); similarly, the *label* attribute on the **<node>** elements records the labels of those nodes. The *order* and *size* attributes on the **<graph>** element record the number of nodes and number of arcs in the graph respectively; these values are optional (since they can be computed from the rest of the graph), but if they are supplied, they must be consistent with the rest of the encoding. They can thus be used to help check that the graph has been encoded and transmitted correctly. The *degree* attribute on the **<node>** elements record the number of arcs that are incident with that node. It is optional (because redundant), but can be used to help in validity checking: if a value is given, it must be consistent with the rest of the information in the graph. Finally, the *from* and *to* attributes on the **<arc>** elements provide pointers to the nodes connected by those arcs. Since the graph is undirected, no directionality is implied by the use of the *from* and *to* attributes; the values of these attributes could be interchanged in each arc without changing the information specified.

The *adj*, *adjFrom*, and *adjTo* attributes of the **<node>** element provide an alternative method of representing unlabeled arcs, their values being pointers to the nodes which are adjacent to or from that node. The *adj* attribute is to be used for undirected graphs, and the *adjFrom* and *adjTo* attributes for directed graphs. It is a semantic error for the directed adjacency attributes to be used in an undirected graph, and vice versa. Here is a markup of the preceding graph, using the *adj* attribute to represent the arcs.

```
<graph type=undirected
       id=cug2
       label='Airline Connections in Southwestern USA'
       rend='LABEL-PLACE bottom center NODE-FRAME none ARC solid line'
       order=5
       size=4>
  <node label=LAX id=LAX degree=2 adj='LVG PHX'>
  <node label=LVG id=LVG degree=2 adj='LAX PHX'>
  <node label=PHX id=PHX degree=3 adj='LAX LVG TUS'>
  <node label=TUS id=TUS degree=1 adj='PHX'>
  <node label=CIB id=CIB degree=0>
  </graph>
```

Note that each arc is represented twice in this encoding of the graph. For example, the existence of the arc from LAX to LVG can be inferred from each of the first two **<node>** elements in the graph. This redundancy, however, is not required: it suffices to describe an arc in any one of the three places it can be described (either adjacent node, or in a separate **<arc>** element). Here is a less redundant representation of the same graph.

```
<graph type=undirected
       id=cug3
       label='Airline Connections in Southwestern USA'
       rend='LABEL-PLACE bottom center NODE-FRAME none ARC solid line'
       order=5
       size=4>
  <node label=LAX id=LAX degree=2 adj='LVG PHX'>
  <node label=LVG id=LVG degree=2 adj='PHX'>
  <node label=PHX id=PHX degree=3 adj='TUS'>
  <node label=TUS id=TUS degree=1>
  <node label=CIB id=CIB degree=0>
  </graph>
```

Although in many cases the **<arc>** element is redundant (since arcs can be described using the adjacency attributes of their adjacent nodes), it has nevertheless been included in the tag set, in order to allow the convenient specification of identifiers, display or rendition information, and labels for each arc (using the attributes *id*, *rend*, and *label*).

Next, let us modify the preceding graph by adding directionality to the arcs. Specifically, we now think of the arcs as specifying selected routes from one airport to another, as indicated by the direction of the arrowhead in the following diagram.

```
 ,-<-LAX-<-,
 |         |
LVG--->---PHX--->---TUS   CIB
          |         |
          '----<----'

Selected Airline Routes in Southwestern USA
```

Here is an encoding of this graph, using the **<arc>** element to designate the arcs.

```
<graph type=directed
       id=rdg1
       label='Selected Airline Routes in Southwestern USA'
       rend='LABEL-PLACE bottom center NODE-FRAME none
             ARC solid line with arrowhead'
       order=5
       size=5>
   <node label=LAX id=LAX inDegree=1 outDegree=1>
   <node label=LVG id=LVG inDegree=1 outDegree=1>
   <node label=PHX id=PHX inDegree=2 outDegree=2>
   <node label=TUS id=TUS inDegree=1 outDegree=1>
   <node label=CIB id=CIB inDegree=0 outDegree=0>
   <arc  from=LAX to=LVG>
   <arc  from=LVG to=PHX>
   <arc  from=PHX to=LAX>
   <arc  from=PHX to=TUS>
   <arc  from=TUS to=PHX>
   </graph>
```

Here is another encoding of the graph, using the *adjTo* and *adjFrom* attributes on nodes to designate the arcs.

```
<graph type=directed
       id=rdg2
       label='Selected Airline Routes in Southwestern USA'
       rend='LABEL-PLACE bottom center NODE-FRAME none ARC solid line
with arrowhead'
       order=5
       size=5>
   <node label=LAX id=LAX inDegree=1 outDegree=1 adjTo=LVG adjFrom=PHX>
   <node label=LVG id=LVG inDegree=1 outDegree=1 adjFrom=LAX adjTo=PHX>
   <node label=PHX id=PHX inDegree=2 outDegree=2 adjTo='LAX TUS'
         adjFrom='LVG TUS'>
   <node label=TUS id=TUS inDegree=1 outDegree=1 adjTo=PHX adjFrom=PHX>
   <node label=CIB id=CIB inDegree=0 outDegree=0>
   </graph>
```

If we wish to label the arcs, say with flight numbers, then **<arc>** elements must be used to carry the *label* attribute, as in the following example.

```
<graph type=directed
       id=rdg1
       label='Selected Routes'
       rend='LABEL-PLACE bottom center NODE-FRAME none ARC solid line
with arrowhead ARC-LABEL-PLACE above'
       order=5
       size=5>
   <node label=LAX id=LAX inDegree=1 outDegree=1>
   <node label=LVG id=LVG inDegree=1 outDegree=1>
   <node label=PHX id=PHX inDegree=2 outDegree=2>
   <node label=TUS id=TUS inDegree=1 outDegree=1>
   <node label=CIB id=CIB inDegree=0 outDegree=0>
   <arc  from=LAX to=LVG label='SW117'>
   <arc  from=LVG to=PHX label='SW711'>
   <arc  from=PHX to=LAX label='AA218'>
   <arc  from=PHX to=TUS label='AW229'>
   <arc  from=TUS to=PHX label='AW225'>
   </graph>
```

The formal declarations of the **<graph>**, **<node>** and **<arc>** elements are as follows.

```
<!-- 23.1:  Graphs                                           -->
<!ELEMENT graph         - -  ((node)+ & (arc)*)                >
<!ATTLIST graph              %a.global;
          type               CDATA             #IMPLIED
          label              CDATA             #IMPLIED
          order              NUMBER            #IMPLIED
          size               NUMBER            #IMPLIED        >
<!ELEMENT node          - O  EMPTY                             >
<!ATTLIST node               %a.global;
          label              CDATA             #IMPLIED
```

```
          label2            CDATA               #IMPLIED
          value             IDREF               #IMPLIED
          type              CDATA               #IMPLIED
          adjTo             IDREFS              #IMPLIED
          adjFrom           IDREFS              #IMPLIED
          adj               IDREFS              #IMPLIED
          inDegree          NUMBER              #IMPLIED
          outDegree         NUMBER              #IMPLIED
          degree            NUMBER              #IMPLIED       >
<!ELEMENT arc          - O  EMPTY                              >
<!ATTLIST arc               %a.global;
          label             CDATA               #IMPLIED
          label2            CDATA               #IMPLIED
          from              IDREF               #REQUIRED
          to                IDREF               #REQUIRED      >
<!-- This fragment is used in sec. 23                        -->
```

23.1.1 Transition Networks

For encoding transition networks and other kinds of directed graphs in which distinctions among types of nodes must be made, the *type* attribute is provided for **<node>** elements. In the following example, the *initial* and *final* nodes (or *states*) of the network are distinguished. It can be understood as accepting the set of strings obtained by traversing it from its initial node to its final node, and concatenating the labels.

```
                 OLD      MAN      COMES
                ,->-,   ,-->----o---->---,
                |   |   |                |
                \   /   |                |
            THE  \ /    |                |
(8)   o-->----o-----+                o
                        |                |
                        |                |
                        |                |
                        '-->----o---->---'
                          MEN       COME
```

```
<graph type='transition network'
       id=ss8
       label='(8)'
       rend='LABEL-PLACE left margin NODE dot ARC solid line with
             arrowhead ARC-LABEL-PLACE outside'
       order=5
       size=6>
   <node id=q0 inDegree=0 outDegree=1 type=initial>
   <node id=q1 inDegree=2 outDegree=3>
   <node id=q2 inDegree=1 outDegree=1>
   <node id=q3 inDegree=1 outDegree=1>
   <node id=q4 inDegree=2 outDegree=0 type=final>
   <arc  from=q0 to=q1 label=THE>
   <arc  from=q1 to=q1 label=OLD>
```

```
<arc  from=q1 to=q2 label=MAN>
<arc  from=q1 to=q3 label=MEN>
<arc  from=q2 to=q4 label=COMES>
<arc  from=q3 to=q4 label=COME>
</graph>
```

A finite transducer has two labels on each arc, and can be thought of as representing a mapping from one sequence of labels to the other. The following example represents a transducer for translating the English strings accepted by the network in the preceding example into French. The nodes have been labeled with numbers, for convenience.

```
         THE         MAN      COMES
      ,--->---1------->-----4--->---,
      |   L'  |     HOMME     VIENT |
      |       |                     |
      |    OLD~VIEIL                |
      |       |                     |
      | THE   |                     |
(8)   0-->----2                     6
      |  LE  / \                    |
      |     /   \                   |
      |    /     \                  |
      |    | OLD |                  |
      |    '-->--'                  |
      |     VIEIL                   |
      |  THE        MEN       COME  |
      '--->---3------>----5---->----'
         LES / \   HOMMES     VONT
            /   \
           /     \
           | OLD |
           '-->--'
            VIEUX
```

```
<graph type='transducer'
       rend='ARC solid line with arrowhead ARC-LABEL-PLACE top
             ARC-LABEL2-PLACE bottom'
       order=7
       size=10>
   <node id=t0 label=0 inDegree=0 outDegree=3 type=initial>
   <node id=t1 label=1 inDegree=2 outDegree=1>
   <node id=t2 label=2 inDegree=2 outDegree=2>
   <node id=t3 label=3 inDegree=2 outDegree=2>
   <node id=t4 label=4 inDegree=1 outDegree=1>
   <node id=t5 label=5 inDegree=1 outDegree=1>
   <node id=t6 label=6 inDegree=2 outDegree=0 type=final>
   <arc  from=t0 to=t1 label=THE label2="L'">
   <arc  from=t0 to=t2 label=THE label2=LE>
   <arc  from=t0 to=t3 label=THE label2=LES>
   <arc  from=t1 to=t4 label=MAN label2=HOMME>
   <arc  from=t2 to=t1 label=OLD label2=VIEIL>
```

```
<arc  from=t2 to=t2 label=OLD label2=VIEIL>
<arc  from=t3 to=t3 label=OLD label2=VIEUX>
<arc  from=t3 to=t5 label=MEN label2=HOMMES>
<arc  from=t4 to=t6 label=COMES label2=VIENT>
<arc  from=t5 to=t6 label=COME label2=VONT>
</graph>
```

23.1.2 Family Trees

The next example provides an encoding a portion of a “family tree”, in which nodes are used to represent individuals, and parents of individuals, and arcs are used to represent common parentage and descent links. Let us suppose, further, that information about individuals is contained in feature structures, which are contained in feature-structure libraries elsewhere in the document (see 18.2, “Feature, Feature Structure, and Feature-Value Libraries,” [in separate fascicle]). We can use the *value* attribute on **<node>** elements to point to those feature structures. Assume that, in the actual graph, nodes representing females are framed by circles, nodes representing males are framed by boxes, and nodes representing parents are framed by diamonds.

```
                            Mo     Fa
                  Katherine->--K+A--<-Amberley
                                |
                        So      |    Da
                   ,----<-------+----->----,
                   |            | So       |
                Bertrand        '->-,    Rachel
                   |                |
                   |              Frank
         Mo     Fa | Fa     Mo
   Peter->--P+B--<-'->--D+B--<-Dora
             |           |
          So |        Da | So
         ,-<-'     ,--<--+-->--,
         |         |           |
       Conrad     Kate        John
```

```
<graph type='family tree' order=13 size=12
       rend='ARC-LABEL-PLACE above'>
  <node id=KathR rend='FRAME circle' label='Katherine'
        value=kr1 inDegree=0 outDegree=1>
  <node id=AmbeR rend='FRAME box' label='Amberley'
        value=ar1 inDegree=0 outDegree=1>
  <node id=KAR rend='FRAME diamond' label='K+A'
        inDegree=2 outDegree=3>
  <node id=BertR rend='FRAME box' label='Bertrand'
        value=br1 inDegree=1 outDegree=2>
  <node id=PeteR rend='FRAME circle' label='Peter'
        value=pr1 inDegree=0 outDegree=1>
  <node id=DoraR rend='FRAME circle' label='Dora'
        value=dr1 inDegree=0 outDegree=1>
  <node id=PBR rend='FRAME diamond' label='P+B'
```

```
          inDegree=2 outDegree=1>
    <node id=DBR rend='FRAME diamond' label='D+B'
          inDegree=2 outDegree=2>
    <node id=FranR rend='FRAME box' label='Frank'
          value=fr1 inDegree=1 outDegree=0>
    <node id=RachR rend='FRAME circle' label='Rachel'
          value=rr1 inDegree=1 outDegree=0>
    <node id=ConrR rend='FRAME box' label='Conrad'
          value=cr1 inDegree=1 outDegree=0>
    <node id=KateR rend='FRAME circle' label='Kate'
          value=kr2 inDegree=1 outDegree=0>
    <node id=JohnR rend='FRAME box' label='John'
          value=jr1 inDegree=1 outDegree=0>
    <arc rend='DRAW arrow' label='Mo' from=KathR to=KAR>
    <arc rend='DRAW arrow' label='Fa' from=AmbeR to=KAR>
    <arc rend='DRAW line'  label='So' from=KAR to=BertR>
    <arc rend='DRAW line'  label='So' from=KAR to=FranR>
    <arc rend='DRAW line'  label='Da' from=KAR to=RachR>
    <arc rend='DRAW arrow' label='Mo' from=PeteR to=PBR>
    <arc rend='DRAW arrow' label='Fa' from=BertR to=PBR>
    <arc rend='DRAW line'  label='So' from=PBR to=ConrR>
    <arc rend='DRAW arrow' label='Mo' from=DoraR to=DBR>
    <arc rend='DRAW arrow' label='Fa' from=BertR to=DBR>
    <arc rend='DRAW line'  label='Da' from=DBR to=KateR>
    <arc rend='DRAW line'  label='So' from=DBR to=JohnR>
    </graph>
  <!-- Partial family tree for Bertrand Russell, based on       -->
  <!-- Pereira and Shieber, Prolog and Natural Language         -->
  <!-- Analysis, 1987, p22                                       -->
```

23.1.3 Historical Interpretation

For our final example, we represent graphically the relationships among various geographic areas mentioned in a seventeenth-century Scottish document. The document itself is a “sasine”, which records a grant of land from the earl of Argyll to one Donald McNeill, and reads in part as follows (abbreviations have been expanded silently, and “[...]” marks illegible passages):

> Item instrument of Sasine given the said Hector Mcneil confirmed and dated 28 May 1632 [...] at Edinburgh upon the 15 June 1632
>
> Item ane charter granted by Archibald late earl of Argyle and Donald McNeill of Gallachalzie wh makes mention that ... the said late Earl yields and grants to the said Donald MacNeill ...
>
> All and hail the two merk land of old extent of Gallachalzie with the pertinents by and in the lordship of Knapdale within the sherrifdome of Argyll
>
> [description of other lands granted follows ...]

This Charter is dated at Inverary the 15th May 1669

In this example, we are concerned with the land and pertinents (i.e., accompanying sources of revenue) described as “the two merk land of old extent of Gallachalzie with the pertinents by and in the lordship of Knapdale within the sherrifdome of Argyll”.

The passage concerns the following pieces of land:

- the Earl of Argyll's land (i.e. the lands granted by this clause of the sasine)
- two mark of land in Gallachalzie
- the pertinents for this land
- the Lordship of Knapdale
- the sherrifdom of Argyll

We will represent these geographic entities as nodes in a graph. Arcs in the graph will represent the following relationships among them:

- containment (INCLUDE)
- location within (IN)
- contiguity (BY)
- constituency (PART OF)

Note that these relationships are logically related: INCLUDE and IN, for example, are inverses of each other: the Earl of Argyll's land INCLUDEs the parcel in Gallachalzie, and the parcel is therefore IN the Earl of Argyll's land. Given an explicit set of inference rules, an appropriate application could use the graph we are constructing to infer the logical consequences of the relationships we identify.

Let us assume that feature-structure analyses are available which describe Gallachalzie, Knapdale, and Argyll. We will link to those feature structures using the *value* attribute on the nodes representing those places. However, there may be some uncertainty as to which noun phrase is modified by the phrase “within the sheriffdome of Argyll”: perhaps the entire lands (land and pertinents) are in Argyll, perhaps just the pertinents are, or perhaps only Knapdale is (together with the portion of the pertinents which is in Knapdale). We will represent all three of these interpretations in the graph; they are, however, mutually exclusive, which we represent using the *excl* attribute defined in chapter 17, “Additional Tag Set for Simple Analysis,” [in separate fascicle].[3]

We represent the graph and its encoding as follows, where the dotted lines in the graph indicate the mutually exclusive arcs; in the encoding, we use the *exclude* attribute to indicate those arcs.

---

[3] That is, the three syntactic interpretations of the clause are mutually exclusive. The notion that the pertinents are in Argyll is clearly not inconsistent with the notion that both the land in Gallachalzie and the pertinents are in Argyll. The graph given here describes the possible interpretations of the clause itself, not the sets of inferences derivable from each syntactic interpretation, for which it would be convenient to use the facilities described in chapter 18, “Additional Tag Set for Feature Structures,” [in separate fascicle].

```
             Earl of Argyll's land
           (land described in sasine)
                     | | |
     ,-<-INCLUDE--<--' + '->-INCLUDE---> Pertinents
     |                 :                  :   |   |
Gallachalzie          IN  ...<..IN..<.....:   |   |
                       :  :                   |   |
                       :  :             INCLUDE INCLUDE
                       :  :              |           |
                       :  :   (part of pertinents)  (part)
                       :  :              |           |
                       :  :              BY         PART OF
                       :  :              |           |
                       :  :          Lordship of Knapdale
                       :  :                   :
                       :  :  ...<.IN..<.......:
                       :  :  :
               Sherrifdome of Argyll
```

The graph formalizes the following relationships:

- the Earl of Argyll's land INCLUDEs (the parcel of land in) Gallachalzie
- the Earl of Argyll's land INCLUDEs the pertinents of that parcel
- the pertinents are (in part) BY the Lordship of Knapdale
- the pertinents are (in part) PART OF the Lordship of Knapdale
- the Earl of Argyll's land, or the pertinents, or the Lordship of Knapdale, is IN the Sherrifdom of Argyll

We encode the graph in SGML thus:

```
<graph type=directed order=7 size=9>
  <node id=earl label="Earl of Argyll's land">
  <node id=gall label="Gallachalzie" value=gallfs>
  <node id=pert label="Pertinents">
  <node id=per1 label="Pertinents part">
  <node id=per2 label="Pertinents part">
  <node id=knap label="Lordship of Knapdale" value=knapfs>
  <node id=argy label="Sherrifdome of Argyll" value=argyfs>
  <arc  id=earlgall label="INCLUDE" from=earl to=gall>
  <arc  id=earlargy label="IN" from=earl to=argy
        exclude="pertargy knapargy">
  <arc  id=earlpert label="INCLUDE" from=earl to=pert>
  <arc  id=pertper1 label="INCLUDE" from=pert to=per1>
  <arc  id=pertper2 label="INCLUDE" from=pert to=per2>
  <arc  id=pertargy label="IN" from=pert to=argy
        exclude="earlargy knapargy">
  <arc  id=per1knap label="BY" from=per1 to=knap>
  <arc  id=per2knap label="PART OF" from=per2 to=knap>
  <arc  id=knapargy label="IN" from=knap to=argy
        exclude="earlargy pertargy">
  </graph>
```

23.2 Trees

A *tree* is a connected acyclic graph. That is, it is possible in a tree graph to follow a path from any vertex to any other vertex, but there are no paths that lead from any vertex to itself. A rooted tree is a directed graph based on a tree; that is, the arcs in the graph correspond to the arcs of a tree such that there is exactly one node, called the *root*, for which there is a path from that node to all other nodes in the graph. For our purposes, we may ignore all trees except for rooted trees, and hence we shall use the **<tree>** element for rooted trees, and the **<root>** element for its root. The nodes adjacent to a given node are called its *children*, and the node adjacent from a given node is called its *parent*. Nodes with both a parent and children are called *internal nodes*, for which we use the **<iNode>** element. A node with no children is tagged as a **<leaf>**. If the children of a node are ordered from left to right, then we say that that node is *ordered*. If all the nodes of a tree are ordered, then we say that the tree is an *ordered tree*. If some of the nodes of a tree are ordered and others are not, then the tree is a *partially ordered tree*. The ordering of nodes and trees may be specified by an attribute; we take the default ordering for trees to be ordered, that roots inherit their ordering from the trees in which they occur, and internal nodes inherit their ordering from their parents. Finally, we permit a node to be specified as following other nodes, which (when its parent is ordered) it would be assumed to precede, giving rise to crossing arcs. The elements used for the encoding of trees have the following descriptions and attributes.

<tree> : encodes a tree, which is made up of a root, internal nodes, leaves, and arcs from root to leaves. Attributes include:
- *arity* : gives the maximum number of children of the root and internal nodes of the tree.
- *ord* : indicates whether or not the tree is ordered, or if it is partially ordered. Legal values are:
  - *Y* : indicates that all of the branching nodes of the tree are ordered.
  - *partial* : indicates that some of the branching nodes of the tree are ordered and some are unordered.
  - *N* : indicates that all of the branching nodes of the tree are unordered.
- *order* : gives the order of the tree, i.e., the number of its nodes.

<root> : represents the root node of a tree. Attributes include:
- *label* : gives a label for a root node.
- *value* : provides the value of the root, which is a feature structure or other analytic element.
- *children* : provides a list of IDs of the elements which are the children of the root node.
- *ord* : indicates whether or not the root is ordered. Legal values are:
  - *Y* : indicates that the children of the root are ordered.
  - *N* : indicates that the children of the root are unordered.
- *outDegree* : gives the out degree of the root, the number of its children.

<iNode> : represents an intermediate (or internal) node of a tree. Attributes include:
- *label* : gives a label for an intermediate node.
- *value* : provides the value of an intermediate node, which is a feature structure or other analytic element.
- *children* : provides a list of IDs of the elements which are the children of the intermediate node.
- *parent* : provides the ID of the element which is the parent of this node.

- *ord* : indicates whether or not the internal node is ordered. Legal values are:
  - *Y* : indicates that the children of the intermediate node are ordered.
  - *N* : indicates that the children of the intermediate node are unordered.
- *follow* : provides an ID of the element which this node follows.
- *outDegree* : gives the out degree of an intermediate node, the number of its children.

<leaf> : encodes the leaves (terminal nodes) of a tree. Attributes include:

- *label* : gives a label for a leaf.
- *value* : provides the value of a leaf, which is a feature structure or other analytic element.
- *parent* : provides the ID of parent of a leaf.
- *follow* : provides an ID of an element which this leaf follows.

Here is an example of a tree. It represents the order in which the operators of addition (symbolized by “+”), exponentiation (symbolized by “**”) and division (symbolized by “/”) are applied in evaluating the arithmetic formula ((a**2)+(b**2))/((a+b)**2). In drawing the graph, the root is placed on the far right, and directionality is presumed to be to the left.

```
a--,
   |-**-,
2--'    |
        |-+--,
b--,    |    |
   |-**-'    |
2--'         |
             |-/
a--,         |
   |-+--,    |
b--'    |    |
        |-**-'
        |
2-------'
```

```
<tree n='ex1' arity=2 order=12>
   <root  label='/'  id=div1              children='plu1 exp1'>
   <iNode label='+'  id=plu1   parent=div1 children='exp2 exp3'>
   <iNode label='**' id=exp1   parent=div1 children='plu2 num2.3'>
   <iNode label='**' id=exp2   parent=plu1 children='vara1 num2.1'>
   <iNode label='**' id=exp3   parent=plu1 children='varb1 num2.2'>
   <iNode label='+'  id=plu2   parent=exp1 children='vara2 varb2'>
   <leaf  label='a'  id=vara1  parent=exp2>
   <leaf  label='2'  id=num2.1 parent=exp2>
   <leaf  label='b'  id=varb1  parent=exp3>
   <leaf  label='2'  id=num2.2 parent=exp3>
   <leaf  label='a'  id=vara2  parent=plu2>
   <leaf  label='b'  id=varb2  parent=plu2>
   <leaf  label='2'  id=num2.3 parent=exp1>
   </tree>
```

In this encoding, the *arity* attribute represents the *arity* of the tree, which is the greatest value of the *outDegree* attribute for any of the nodes in the tree. If, as in this case, *arity=2*, we say that the tree is a *binary* tree.

Since the left-to-right (or top-to-bottom!) order of the children of the two "+" nodes does not affect the arithmetic result in this case, we could represent in this tree all of the arithmetically equivalent formulas involving its leaves, by specifying the value *ord* = N on those two **<iNode>** elements, the value *ord* = Y on the **<root>** and other **<iNode>** elements, and the value *ord* = partial on the **<tree>** element, as follows.

```
<tree n='ex2' ord=partial arity=2 order=13>
 <root  label='/'  id=div1   ord=Y            children='plu1 exp1'>
 <iNode label='+'  id=plu1   ord=N parent=div1 children='exp2 exp3'>
 <iNode label='**' id=exp1   ord=Y parent=div1 children='plu2 num2.3'>
 <iNode label='**' id=exp2   ord=Y parent=plu1 children='vara1 num2.1'>
 <iNode label='**' id=exp3   ord=Y parent=plu1 children='varb1 num2.2'>
 <iNode label='+'  id=plu2   ord=N parent=exp1 children='vara2 varb2'>
 <leaf  label='a'  id=vara1        parent=exp2>
 <leaf  label='2'  id=num2.1       parent=exp2>
 <leaf  label='b'  id=varb1        parent=exp3>
 <leaf  label='2'  id=num2.2       parent=exp3>
 <leaf  label='a'  id=vara2        parent=plu2>
 <leaf  label='b'  id=varb2        parent=plu2>
 <leaf  label='2'  id=num2.3       parent=exp1>
 </tree>
```

This encoding represents all of the following:

- ((a**2)+(b**2))/((a+b)**2)
- ((b**2)+(a**2))/((a+b)**2)
- ((a**2)+(b**2))/((b+a)**2)
- ((b**2)+(a**2))/((a+b)**2)

Linguistic phrase structure is very commonly represented by trees. Here is an example of phrase structure represented by an ordered tree with its root at the top, and a possible encoding.

```
  ,---PP---,
  |        |
  P    ,--NP--,
  |    |      |
 with Art     N
       |      |
      the periscope
```

```
<tree n='ex3' arity=2 order=8>
   <root  id=PP1             children='P1 NP1'     label='PP'>
   <iNode id=P1    parent=PP1  children=with1       label='P'>
   <leaf  id=with1 parent=P1                        label='with'>
   <iNode id=NP1   parent=PP1  children='the1 peri1' label='NP'>
   <iNode id=Art1  parent=NP1  children=the1        label='Art'>
   <leaf  id=the1  parent=Art1                      label='the'>
   <iNode id=N1    parent=NP1  children=peri1       label='N'>
   <leaf  id=peri1 parent=N1                        label='periscope'>
   </tree>
```

Finally, here is an example of an ordered tree, in which a particular node which ordinarily would precede another is specified as following it. In the drawing, the “~~~” symbol indicates that the arc from VB to PT crosses the arc from VP to PN.

```
         ,--VP--,
         |      |
     ,--VB-----~~~--,
     |          |   |
     VB         PN  PT
     |          |   |
    look       them up

    <tree n='ex4' arity=2 order=8>
       <leaf  label='look' id=look1 parent=VB2>
       <leaf  label='them' id=them1 parent=PN1>
       <leaf  label='up'   id=up1   parent=PT1>
       <iNode label='VB'   id=VB2   parent=VB1 children=look1>
       <iNode label='PN'   id=PN1   parent=VP1 children=them1 >
       <iNode label='PT'   id=PT1   parent=VB1 children=up1 follow=PN1>
       <iNode label='VB'   id=VB1   parent=VP1 children='VB2 PT1'>
       <root  label='VP'   id=VP1              children='VB1 PN1'>
       </tree>
```

The formal declarations of the **<tree>**, **<root>**, **<iNode>** and **<leaf>** elements are as follows.

```
<!-- 23.2:  Trees (basic method)                              -->
<!ELEMENT tree          - -  ((leaf | iNode)*, root, (leaf |
                             iNode)*)                          >
<!ATTLIST tree               %a.global;
          label              CDATA              #IMPLIED
          arity              NUMBER             #IMPLIED
          ord                (Y | N | partial)   Y
          order              NUMBER             #IMPLIED       >
<!ELEMENT root          - O  EMPTY                             >
<!ATTLIST root               %a.global;
          label              CDATA              #IMPLIED
          value              IDREF              #IMPLIED
          children           IDREFS             #IMPLIED
          ord                (Y | N)            #IMPLIED
          outDegree          NUMBER             #IMPLIED       >
<!ELEMENT iNode         - O  EMPTY                             >
<!ATTLIST iNode              %a.global;
          label              CDATA              #IMPLIED
          value              IDREF              #IMPLIED
          children           IDREFS             #REQUIRED
          parent             IDREF              #IMPLIED
          ord                (Y | N)            #IMPLIED
          follow             IDREF              #IMPLIED
          outDegree          NUMBER             #IMPLIED       >
<!ELEMENT leaf          - O  EMPTY                             >
<!ATTLIST leaf               %a.global;
          label              CDATA              #IMPLIED
```

```
          value           IDREF             #IMPLIED
          parent          IDREF             #IMPLIED
          follow          IDREF             #IMPLIED       >
<!-- This fragment is used in sec. 23                       -->
```

23.3 Another Tree Notation

In this section, we present an alternative to the method of representing the structure of ordered rooted trees that is given in section 23.2, "Trees," on page 15, which is based on the observation that any node of such a tree can be thought of as the root of the subtree that it dominates. Thus subtrees can be thought of as the same type as the trees they are embedded in, hence the designation **<eTree>**, for *embedding tree*. Whereas in a **<tree>**, the relationship among the parts is indicated by the *children* attribute, and by the names of the elements **<root>**, **<iNode>** and **<leaf>**, the relationship among the parts of an **<eTree>** is indicated simply by the arrangement of their content. However, we have chosen to enable encoders to distinguish the terminal elements of an **<eTree>** by means of the empty **<eLeaf>** element, though its use is not required; the **<eTree>** element can also be used to identify the terminal nodes of **<eTree>** elements. We also provide a **<triangle>** element, which can be thought of as an *underspecified* **<eTree>**, that is an **<eTree>** in which certain information has been left out. In addition, we provide a **<forest>** element, which consists of one or more **<tree>**, **<eTree>** or **<triangle>** elements, and a **<forestGrp>** element, which consists of one or more **<forest>** elements. The elements used for the encoding of embedding trees and the units containing them have the following descriptions and attributes.

<eTree> : provides an alternative to **<tree>** element for representing ordered rooted tree structures. Attributes include:
- *label* : gives a label for an embedding tree.
- *value* : provides the value of an embedding tree, which is a feature structure or other analytic element.

<triangle> : provides for an underspecified **<eTree>**, that is, an **<eTree>** with information left out. Attributes include:
- *label* : gives a label for an underspecified embedding tree.
- *value* : provides the value of a triangle, which is the SGML identifier of a feature structure or other analytic element.

<eLeaf> : provides explicitly for a leaf of an embedding tree, which may also be encoded with the **<eTree>** element. Attributes include:
- *label* : gives a label for a leaf of an embedding tree.
- *value* : provides the value of an embedding leaf, which is a feature structure or other analytic element.

<forest> : provides for groups of rooted trees. Attributes include:
- *type* : identifies the type of the forest.

<forestGrp> : provides for groups of forests. Attributes include:
- *type* : identifies the type of the forest group.

Like the **<root>**, **<iNode>** and **<leaf>** of a **<tree>**, the **<eTree>**, **<triangle>** and **<eLeaf>** elements may also have *label* and *value* attributes.

To illustrate the use of the **<eTree>** and **<eLeaf>** elements, here is an encoding of the second example in section 23.2, "Trees," on page 15, repeated here for convenience.

```
 ,---PP---,
 |        |
 P    ,--NP--,
 |    |      |
with Art     N
      |      |
     the periscope
```

```
<eTree n='ex1' label='PP'>
   <eTree label='P'><eLeaf label='with'></eTree>
   <eTree label='NP'>
      <eTree label='Art'><eLeaf label='the'></eTree>
      <eTree label='N'><eLeaf label='periscope'></eTree>
      </eTree>
   </eTree>
```

Next, we provide an encoding, using the **<triangle>** element, in which the internal structure of the **<eTree>** labeled “NP” is omitted.

```
 ,---PP----,
 |         |
 P        NP
 |       /  \
with    /    \
       /______\
     the periscope
```

```
<eTree n='ex2' label='PP'>
   <eTree label='P'><eLeaf label='with'></eTree>
   <triangle label='NP'><eLeaf label='the periscope'></triangle>
   </eTree>
```

Ambiguity involving alternative tree structures associated with the same terminal sequence can be encoded relatively conveniently using a combination of the *exclude* and *copyOf* attributes described in sections 16.8, “Alternations,” [in separate fascicle] and 16.6, “Identical Elements and Virtual Copies,” [in separate fascicle]. In the simplest case, an **<eTree>** may be part of the content of exactly one of two different **<eTree>** elements. To mark it up, the embedded **<eTree>** may be fully specified within one of the embedding **<eTree>** elements to which it may belong, and a virtual copy, specified by the *copyOf* attribute, may appear on the other. In addition, each of the embedded elements in question is specified as excluding the other, using the *exclude* attribute. To illustrate, consider the English phrase “see the vessel with the periscope”, which may be considered to be structurally ambiguous, depending on whether the phrase “with the periscope” is a modifier of the phrase “the vessel” or a modifier of the phrase “see the vessel”. This ambiguity is indicated in the sketch of the ambiguous tree by means of the dotted-line arcs. The markup using the *copyOf* and *exclude* attributes follows the sketch.

```
 ,--------VP...................
 |        |                    :
 V   ,----NP.......         ,--PP--,
 |   |    |       :         |      |
see Art   N    ,--PP--,     P    ,-NP-,
```

```
          |    |    |      |     |    |    |
         the vessel P    ,-NP-, with Art   N
                    |    |    |       |    |
                   with Art   N      the periscope
                         |    |
                        the periscope

<eTree n='ex3' label='VP'>
   <eTree label='V'><eLeaf label='see'></eTree>
   <eTree label='NP'>
      <eTree label='Art'><eLeaf label='the'></eTree>
      <eTree label='N'><eLeaf label='vessel'></eTree>
      <eTree id=ppa exclude=ppb label='PP'>
         <eTree label='P'><eLeaf label='with'></eTree>
         <eTree label='NP'>
            <eTree label='Art'><eLeaf label='the'></eTree>
            <eTree label='N'><eLeaf label='periscope'></eTree>
            </eTree>
         </eTree>
      </eTree>
   <eTree id=ppb copyOf=ppa exclude=ppa label='PP'></eTree>
   </eTree>
```

To indicate that one of the alternatives is selected, one may specify the *select* attribute on the highest **<eTree>** as either *select=ppa* or *select=ppb*; see section 16.8, “Alternations,” [in separate fascicle].

Depending on the grammar one uses to associate structures with examples like *see the man with the periscope*, the representations may be more complicated than this. For example, adopting a version of the *X-bar* theory of phrase structure originated by Ray Jackendoff in *X-Bar Syntax*, 1977, the attachment of a modifier may require the creation of an intermediate node which is not required when the attachment is not made, as shown in the following diagram. A possible encoding of this ambiguous structure immediately follows the diagram.

```
    ..............VP......................
    :                                    :
 ,--V'--,                          ,-----V'------,
 |      |                          |             |
 V    ,-NP--,                   ,--V'--,      ,--PP--,
 |    |     |                   |      |      |      |
see  Spec ,-N'--,               V    ,-NP-,   P    ,-NP-,
      |   |     |               |    |    |   |    |    |
     the  N'  ,-PP---,         see  Spec  N' with Spec  N'
          |   |      |               |    |        |    |
          N   P    ,-NP-,           the   N       the   N
          |   |    |    |                 |             |
      vessel with Spec  N'             vessel        periscope
                   |    |
                  the   N
                        |
                     periscope
```

```
<eTree n='ex4' label="VP">
   <eTree id=vbara exclude=vbarb label="V'">
      <eTree id=va label="V"><eLeaf label="see"></eTree>
      <eTree label="NP">
         <eTree id=spec1a label="Spec"><eLeaf label="the"></eTree>
         <eTree label="N'">
            <eTree id=nbar2a label="N'">
               <eTree label="N"><eLeaf label="vessel"></eTree>
               </eTree>
            <eTree id=ppa label="PP">
               <eTree label="P"><eLeaf label="with"></eTree>
               <eTree label="NP">
                  <eTree label="Spec"><eLeaf label="the"></eTree>
                  <eTree label="N'">
                     <eTree label="N"><eLeaf label="periscope"></eTree>
                     </eTree>
                  </eTree>
               </eTree>
            </eTree>
         </eTree>
      </eTree>
   <eTree id=vbara exclude=vbarb label="V'">
      <eTree label="V'">
         <eTree id=vb copyOf=va label="V"></eTree>
         <eTree label="NP">
            <eTree id=spec1b copyOf=spec1a label="Spec"></eTree>
            <eTree id=nbar2b copyOf=nbar2a label="N'"></eTree>
            </eTree>
         </eTree>
      <eTree id=ppb copyOf=ppa label="PP"></eTree>
      </eTree>
   </eTree>
```

A *derivation* in a generative grammar is often thought of as a set of trees. To encode such a derivation, one may use the **<forest>** element, in which the trees may be marked up using the **<tree>**, the **<eTree>** or the **<triangle>** element. The *type* attribute may be used to specify what kind of derivation it is. Here is an example of a two-tree forest, involving application of the “wh-movement” transformation in the derivation of “what you do” (as in “this is what you do”) from the underlying “you do what”.[4]

```
     ,--S'--,                ,--S'--,
     |      |                |      |
    COMP ,--S--,            COMP ,--S--,
     |   |     |             |   |     |
     e   NP  ,-VP-,          NP  NP  ,-VP-,
         |   |    |          |   |   |    |
        you  V    NP       what you  V    NP
             |    |                  |    |
```

&@fn@sep

[4] The symbols “e” and “t” denote special theoretical constructs (*empty category* and *trace* respectively), and need not concern us here.

```
                  do   what                do    t

      <forest n='ex5' type='syntactic derivation'>
         <eTree n='Stage 1' id=s1sbar label="S'">
            <eTree id=s1comp label="COMP"><eleaf id=s1e label="e"></eTree>
            <eTree id=s1s label="S">
               <eTree id=s1np1 label="NP"><eLeaf label="you"></eTree>
               <eTree id=s1vp label="VP">
                  <eTree id=s1v label="V"><eLeaf label="do"></eTree>
                  <eTree id=s1np2 label="NP">
                     <eLeaf id=s1wh label="what">
                     </eTree>
                  </eTree>
               </eTree>
            </eTree>
         <eTree n='Stage 2' id=s2sbar corresp=s1sbar label="S'">
            <eTree id=s2comp corresp=s1comp label="COMP">
               <eTree copyOf=s1np2 corresp=s1e label="NP"></eTree>
               </eTree>
            <eTree id=s2s corresp=s1s label="S">
               <eTree id=s2np1 copyOf=s1np1 label="NP"></eTree>
               <eTree id=s2vp corresp=s1vp label="VP">
                  <eTree id=s2v copyOf=s1v label="V"></eTree>
                  <eTree id=s2np2 corresp=s1np2 label="NP">
                    <eLeaf corresp=s1wh label="t">
                    </eTree>
                  </eTree>
               </eTree>
            </eTree>
         </forest>
```

In this markup, we have used *copyOf* attributes to provide virtual copies of elements in the tree representing the second stage of the derivation that also occur in the first stage, and the *corresp* attribute (see section 16.4, “Correspondence,” [in separate fascicle]) to link those elements in the second stage with corresponding elements in the first stage that are not copies of them.

If a group of forests (e.g. a full grammatical derivation including syntactic, semantic and phonological subderivations) is to be articulated, the grouping element **<forestGrp>** may be used.

The formal declarations of the **<eTree>**, **<triangle>**, **<eLeaf>**, **<forest>** and **<forestGrp>** elements are as follows.

```
      <!-- 23.3:  Trees (alternate method)                        -->
      <!ELEMENT eTree        - -  ((eTree | triangle | eLeaf )*)     >
      <!ATTLIST eTree             %a.global;
                label             CDATA              #IMPLIED
                value             IDREF              #IMPLIED        >
      <!ELEMENT triangle     - -  ((eTree | triangle | eLeaf)*)      >
      <!ATTLIST triangle          %a.global;
                label             CDATA              #IMPLIED
                value             IDREF              #IMPLIED        >
```

```
<!ELEMENT eLeaf         - O  EMPTY                               >
<!ATTLIST eLeaf              %a.global;
          label              CDATA               #IMPLIED
          value              IDREF               #IMPLIED        >
<!ELEMENT forest        - -  ((tree | eTree | triangle)+)        >
<!ATTLIST forest             %a.global;
          type               CDATA               #IMPLIED        >
<!ELEMENT forestGrp     - -  ((forest)+)                         >
<!ATTLIST forestGrp          %a.global;
          type               CDATA               #IMPLIED        >
<!-- This fragment is used in sec. 23                           -->
```

Part 7

Alphabetic Reference List of Tags

Class Name: **analysis**

Description: defines additional analytic attributes specifying alternate structural arrangements for the components of a text.

Attributes:

equiv (i.e. equivalent): specifies the identifiers of elements considered structurally equivalent to the one bearing the attribute.
Data type: IDREFS
Value: one or more valid SGML identifiers for elements in the current document separated by white space.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

exclude: specifies the identifiers of elements that are considered excluded from the structure when the element bearing this attribute is selected.
Data type: IDREFS
Value: one or more valid SGML identifiers for elements in the current document separated by white space.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

select: specifies which of several mutually exclusive elements actually occurs in a structure.
Data type: IDREFS
Value: one or more valid SGML identifiers for elements in the current document separated by white space.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

Part: additional tag set for analysis and interpretation.
Member of classes:
DTD file:

Discussed in 17, “Additional Tag Set for Simple Analysis,” [in separate fascicle].

<arc>

Description: encodes an arc, the connection from one node to another in a graph.
Attributes:

label: gives a label for an arc.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

label2: gives a second label for an arc.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
Remarks: Use this attribute together with the *label* attribute if a transducer is being encoded.
This attribute is optional.

from: gives the ID of the node which is adjacent from this arc.
Data type: IDREF
Value: The ID of a node.
Default value: #REQUIRED
This attribute is required.

to: gives the ID of the node which is adjacent to this arc.
Data type: IDREF
Value: The ID of a node.
Default value: #REQUIRED
This attribute is required.

Example:

```
<arc from=t3 to=t3 label='OLD' label2='VIEUX'>
```

Remarks: The **<arc>** element must be used if the arcs are labeled. Otherwise, arcs can be encoded using the *adj*, *adjTo* and *adjFrom* attributes on the **<node>** tags in the graph. Both **<arc>** tags and adjacency attributes can be used, but the resulting encoding would be highly redundant.
Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: empty
May occur within: graph
May contain: EMPTY
Declarations:

```
<!ELEMENT arc           - O  EMPTY                        >
```

```
<!ATTLIST arc               %a.global;
          label             CDATA              #IMPLIED
          label2            CDATA              #IMPLIED
          from              IDREF              #REQUIRED
          to                IDREF              #REQUIRED      >
```

Discussed in 23, “Graphs, Networks, and Trees,” on page 3.
This tag is optional.

Class Name: **certainty**

Description: defines additional attributes which are enabled for all elements when the additional tag set for certainty and uncertainty is selected.
Attributes: [None: global and inherited attributes only.]
Part: additional tag set for certainty.
Member of classes:
DTD file:
Discussed in 19, “CE: Indications of Certainty and Responsibility,” [in separate fascicle].

Class Name: **chunk**

Description: includes all elements which can occur between, but not within, paragraphs and other chunks.
Attributes: [None: global and inherited attributes only.]
Remarks: This element class does not contain all those elements which can appear between chunks: the class *inter* contains a set of elements which can appear either within or between chunks. Unlike elements of that class, chunks cannot occur within chunks. In prose, this means the elements in this class can appear between but not within paragraphs.
Part: base tag set for common core features.
Member of classes: common
DTD file:
Discussed in 3.7, “Element Classes,” [in separate fascicle].

Class Name: **common**

Description: common chunk- and inter-level elements.
Attributes: [None: global and inherited attributes only.]
Remarks: This class defines the set of chunk- and inter-level elements available in all bases; it is used in defining the standard models *chunk.seq* and *specialPara* in the general and mixed bases.
Part: base tag set for common core features.
Member of classes:

DTD file:
Discussed in 3.7, “Element Classes,” [in separate fascicle].

<eLeaf> (i.e. Leaf of an embedding tree.)

Description: provides explicitly for a leaf of an embedding tree, which may also be encoded with the **<eTree>** element.
Attributes:

label: gives a label for a leaf of an embedding tree.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

value: provides the value of an embedding leaf, which is a feature structure or other analytic element.
Data type: IDREF
Value: A valid identifier of a feature structure or other analytic element.
Default value: #IMPLIED
This attribute is recommended when applicable.

Example:

```
<eLeaf label='with' value=fswith>
```

Remarks: The **<eTree>** tag may be used if the encoder does not wish to distinguish by name between nonleaf and leaf nodes in embedding trees; they are distinguished by their arrangement.
Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: empty
May occur within: eTree triangle
May contain: EMPTY
Declarations:

```
<!ELEMENT eLeaf         - O  EMPTY                          >
<!ATTLIST eLeaf              %a.global;
          label              CDATA              #IMPLIED
          value              IDREF              #IMPLIED    >
```

Discussed in 23.3, “Another Tree Notation,” on page 19.
This tag is optional.

<eTree> (i.e. embedding tree)

Description: provides an alternative to **<tree>** element for representing ordered rooted tree structures.
Attributes:

label: gives a label for an embedding tree.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is recommended when applicable.

value: provides the value of an embedding tree, which is a feature structure or other analytic element.
Data type: IDREF
Value: A valid identifier of a feature structure or other analytic element.
Default value: #IMPLIED
This attribute is recommended when applicable.

Example:

```
<eTree n='ex1' label=PP>
   <eTree label=P><eLeaf label=with></eTree>
   <eTree label=NP>
      <eTree label=Art><eLeaf label=the></eTree>
      <eTree label=N><eLeaf label=periscope></eTree>
      </eTree>
   </eTree>
```

Part: additional tag set for graphs, networks, and trees.
Member of classes: chunk
DTD file:
Data description: zero or more embedding trees, triangles, or embedding leafs.
May occur within: eTree forest triangle
May contain: eLeaf eTree triangle
Declarations:

```
<!ELEMENT eTree         - -  ((eTree | triangle | eLeaf )*)     >
<!ATTLIST eTree              %a.global;
          label              CDATA               #IMPLIED
          value              IDREF               #IMPLIED       >
```

Discussed in 23.3, “Another Tree Notation,” on page 19.
This tag is optional.

<forest>

Description: provides for groups of rooted trees.
Attributes:

type: identifies the type of the forest.
Data type: CDATA

Value: A character string.
Default value: #IMPLIED
This attribute is optional.

Example:

Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: One or more trees, embedding trees, or underspecified embedding trees (triangles).
May occur within: forestGrp
May contain: eTree tree triangle
Declarations:

```
<!ELEMENT forest        - -  ((tree | eTree | triangle)+)       >
<!ATTLIST forest             %a.global;
          type               CDATA              #IMPLIED       >
```

Discussed in 23.3, “Another Tree Notation,” on page 19.
This tag is optional.

<forestGrp> (i.e. Forest group)

Description: provides for groups of forests.
Attributes:

type: identifies the type of the forest group.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

Example:

```
<!--None provided. -->
```

Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: One or more forests.
May occur within:
May contain: forest
Declarations:

```
<!ELEMENT forestGrp     - -  ((forest)+)                        >
<!ATTLIST forestGrp          %a.global;
          type               CDATA              #IMPLIED       >
```

Discussed in 23.3, “Another Tree Notation,” on page 19.
This tag is optional.

Class Name: **fs** (i.e. feature-structures)
Description: defines additional attributes for all elements, which are enabled when the additional tag set for feature-structure notation is selected.
Attributes: [None: global and inherited attributes only.]
Remarks: [No attributes declared yet.]
Part: additional tag set for analysis and interpretation.
Member of classes:
DTD file:
Discussed in 18, “Additional Tag Set for Feature Structures,” [in separate fascicle].

Class Name: **global**
Description: defines attributes common to all elements in the TEI encoding scheme.
Attributes:

id (i.e. identifier): provides a unique identifier for the element bearing the ID value.
Data type: ID
Value: any valid SGML name (in the reference concrete syntax, a name must begin with a letter and contain only letters, numeric digits, hyphen, and full stop).
Default value: #IMPLIED
Example:

```
<p id=names>Paragraph with the ID <mentioned>names</>....
<p id=dates>Paragraph with the ID <mentioned>dates</>....
```

Remarks: The *id* attribute may be used to specify a canonical reference for an element; see section 6.9, “Reference Systems,” [in separate fascicle].
This attribute is optional.

n (i.e. number, name, etc.): gives a number (or other label) for an element, which is not necessarily unique within the document.
Data type: CDATA
Value: any string of characters; often, but not necessarily, numeric.
Default value: #IMPLIED
Example:

Remarks: The *n* attribute may be used to specify the numbering of chapters, sections, list items, etc.; it may also be used in the specification of a standard reference system for the text.
This attribute is optional.

lang (i.e. language): indicates the language of the element content, usually using a two- or three-letter code from ISO 639.
Data type: IDREF
Value: The value must be the identifier specified for a writing system declaration declared in the TEI header, as described in section 5, "The TEI Header," [in separate fascicle].
Default value: %INHERITED
Example:

```
<p lang=EN>The only surviving work by <person>Ari</>
(died 1148) is the ten-page <title
lang=IS>&Iacute;slendingab&oacute;k</> (<title
lang=LA>Libellus Islandorum</>), written in the early
twelfth century.
```

Remarks: If no value is specified for *lang*, the *lang* value for the immediately enclosing element is inherited; for this reason, a value should always be specified on the outermost element (**<tei.2>**).
This attribute is optional.

rend (i.e. rendition or presentation): indicates how the element in question was rendered or presented in the source text.
Data type: CDATA
Value: any string of characters; if the typographic rendition of a text is to be systematically recorded, a systematic set of values for the *rend* attribute should be defined.
Default value: #IMPLIED
Example:

Remarks: These Guidelines make no binding recommendations for the values of the *rend* attribute; the characteristics of visual presentation vary too much from text to text and the decision to record or ignore individual characteristics varies too much from project to project. Some potentially useful conventions are noted from time to time at appropriate points in the Guidelines.
This attribute is optional.

Remarks: The global attributes described here are made part of the attribute definition list declaration of each element by including the string "%a.global" in each such declaration. Some global attributes are made available when certain base or additional tag sets are selected; these are incorporated into the global attributes by references to the appropriate parameter entities. When the tag sets in question have not been selected, the parameter entities in question expand to the empty string.
Part: base tag set for common core features.
Member of classes: clterm cllink clana clfs clcert clspoken

DTD file:
Discussed in 3.5, “Global Attributes,” [in separate fascicle].

<graph>

Description: encodes a graph, which is a collection of nodes, and arcs which connect the nodes.

Attributes:

type: describes the type of graph.
Data type: CDATA
Suggested values include:
undirected: undirected graph
directed: directed graph
transition network: a directed graph with distinguished initial and final nodes
transducer: a transition network with up to two labels on each arc
Default value: #IMPLIED
Remarks: If *type=undirected*, then the distinction between the *to* and *from* attributes of the **<arc>** tag is neutralized. Also, the *adj* attribute, rather than the *adjFrom* and *adjTo* attributes, should be used to encode pointers to the ends of the arcs. If *type=directed* (or any other value which implies directionality), then the *adjFrom* and *adjTo* attributes should be used, instead of the *adj* attribute.
This attribute is recommended.

label: gives a label for a graph.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is recommended when applicable.

order: states the order of the graph, i.e., the number of its nodes.
Data type: NUMBER
Value: A positive integer.
Default value: #IMPLIED
This attribute is optional.

size: states the size of the graph, i.e., the number of its arcs.
Data type: NUMBER
Value: A nonnegative integer.
Default value: #IMPLIED
This attribute is optional.

Example:

```
<graph type=undirected
       id=cug1
       label='Airline Connections in Southwestern USA'
       rend='LABEL-PLACE bottom center NODE-FRAME none ARC solid line'
       order=5
       size=4>
  <node label=LAX id=LAX degree=2>
  <node label=LVG id=LVG degree=2>
  <node label=PHX id=PHX degree=3>
  <node label=TUS id=TUS degree=1>
  <node label=CIB id=CIB degree=0>
  <arc  from=LAX to=LVG>
  <arc  from=LAX to=PHX>
  <arc  from=LVG to=PHX>
  <arc  from=PHX to=TUS>
  </graph>

<graph type=directed
       id=rdg2
       label='Selected Airline Routes in Southwestern USA'
       rend='LABEL-PLACE bottom center NODE-FRAME none ARC solid line
with arrowhead'
       order=5
       size=5>
   <node label=LAX id=LAX inDegree=1 outDegree=1 adjTo=LVG adjFrom=PHX>
   <node label=LVG id=LVG inDegree=1 outDegree=1 adjFrom=LAX adjTo=PHX>
   <node label=PHX id=PHX inDegree=2 outDegree=2 adjTo='LAX TUS'
         adjFrom='LVG TUS'>
   <node label=TUS id=TUS inDegree=1 outDegree=1 adjTo=PHX adjFrom=PHX>
   <node label=CIB id=CIB inDegree=0 outDegree=0>
   </graph>
```

Part: additional tag set for graphs, networks, and trees.
Member of classes: chunk
DTD file:
Data description: One or more nodes and zero or more arcs in any order.
May occur within:
May contain: arc node
Declarations:

```
<!ELEMENT graph         - -  ((node)+ & (arc)*)                >
<!ATTLIST graph              %a.global;
          type               CDATA              #IMPLIED
          label              CDATA              #IMPLIED
          order              NUMBER             #IMPLIED
          size               NUMBER             #IMPLIED       >
```

Discussed in 23, “Graphs, Networks, and Trees,” on page 3.
This tag is recommended when applicable.

<iNode> (i.e. intermediate (or internal) node)

Description: represents an intermediate (or internal) node of a tree.
Attributes:

label: gives a label for an intermediate node.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

value: provides the value of an intermediate node, which is a feature structure or other analytic element.
Data type: IDREF
Value: A valid identifier of a feature structure or other analytic element.
Default value: #IMPLIED
This attribute is recommended when applicable.

children: provides a list of IDs of the elements which are the children of the intermediate node.
Data type: IDREFS
Value: A list of IDs.
Default value: #REQUIRED
This attribute is required.

parent: provides the ID of the element which is the parent of this node.
Data type: IDREF
Value: The ID of the parent node.
Default value: #IMPLIED
This attribute is optional.

ord: indicates whether or not the internal node is ordered.
Data type: (Y | N)
Legal values are:
Y: indicates that the children of the intermediate node are ordered.
N: indicates that the children of the intermediate node are unordered.
Default value: #IMPLIED
Remarks: Use if and only if *ord=partial* is specified on the **<tree>** tag and the intermediate node has more than one child.
This attribute is optional.

follow: provides an ID of the element which this node follows.
Data type: IDREF
Value: An ID of another intermediate node or leaf of the tree.
Default value: #IMPLIED
Remarks: If the tree is unordered or partially ordered, this attribute has the property of fixing the relative order of the intermediate node and the element which is the value of the attribute.
This attribute is recommended when applicable.

outDegree: gives the out degree of an intermediate node, the number of its children.
Data type: NUMBER

Value: A nonnegative integer.
Default value: #IMPLIED
Remarks: The in degree of an intermediate node is always 1.
This attribute is optional.

Example:

```
<iNode label=PT id=PT1 parent=VB1 children=up1 outDegree=1 follow=PN1>
```

Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: empty
May occur within: tree
May contain: EMPTY
Declarations:

```
<!ELEMENT iNode         - O  EMPTY                           >
<!ATTLIST iNode              %a.global;
          label              CDATA              #IMPLIED
          value              IDREF              #IMPLIED
          children           IDREFS             #REQUIRED
          parent             IDREF              #IMPLIED
          ord                (Y | N)            #IMPLIED
          follow             IDREF              #IMPLIED
          outDegree          NUMBER             #IMPLIED       >
```

Discussed in 23.2, “Trees,” on page 15.
This tag is recommended when applicable.

<leaf>

Description: encodes the leaves (terminal nodes) of a tree.
Attributes:

label: gives a label for a leaf.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

value: provides the value of a leaf, which is a feature structure or other analytic element.
Data type: IDREF
Value: A valid identifier of a feature structure or other analytic element.
Default value: #IMPLIED
This attribute is recommended when applicable.

parent: provides the ID of parent of a leaf.
Data type: IDREF

Value: The ID of the parent node.
Default value: #IMPLIED
This attribute is optional.

follow: provides an ID of an element which this leaf follows.
Data type: IDREF
Value: An ID of another intermediate node or leaf of the tree.
Default value: #IMPLIED
Remarks: If the tree is unordered or partially ordered, this attribute has the property of fixing the relative order of the leaf and the element which is the value of the attribute.
This attribute is recommended when applicable.

Example:

```
<leaf id=peri1 parent=N1 label=periscope>
```

Remarks: The in degree of a leaf is always 1, its out degree always 0.
Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: empty
May occur within: tree
May contain: EMPTY
Declarations:

```
<!ELEMENT leaf          - O  EMPTY                                >
<!ATTLIST leaf               %a.global;
          label              CDATA              #IMPLIED
          value              IDREF              #IMPLIED
          parent             IDREF              #IMPLIED
          follow             IDREF              #IMPLIED       >
```

Discussed in 23.2, “Trees,” on page 15.
This tag is recommended when applicable.

Class Name: **linking**
Description: defines additional attributes for hypertext and other linking, which are enabled for all elements when the additional tag set for linking is selected.
Attributes:

corresp (i.e. correspondents): specifies the identifiers of corresponding elements.
Data type: IDREFS
Value: one or more valid SGML identifiers for elements in the current document separated by white space.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

link (i.e. link): specifies the identifiers of elements that may be linked with this one to form an aggregate.
Data type: IDREFS
Value: one or more valid SGML identifiers for elements in the current document separated by white space.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

partof: specifies the identifier of a **<join>** element in the current document representing an aggregate of which this element is a component part.
Data type: IDREF
Value: a valid SGML identifier attached to a **<join>** element.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

prev (i.e. previous): specifies the identifier of an element which may be linked with the current element to form an aggregate, and which immediately precedes it in the aggregate.
Data type: IDREF
Value: a valid SGML identifier for an element in the current document.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

next (i.e. next): specifies the identifier of an element which may be linked with the current element to form an aggregate, and which immediately follows it in the aggregate.
Data type: IDREF
Value: a valid SGML identifier for an element in the current document.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

Part: additional tag set for analysis and interpretation.
Member of classes:

DTD file:
Discussed in 16, “Segmentation and Alignment,” [in separate fascicle].

<node>

Description: encodes a node, a possibly labeled point in a graph.
Attributes:

label: gives a label for a node.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

label2: gives a second label for a node.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
Remarks: Use this attribute together with the *label* attribute if a transducer is being encoded whose actions are associated with nodes rather than with arcs.
This attribute is optional.

value: provides the value of a node, which is a feature structure or other analytic element.
Data type: IDREF
Value: A valid identifier.
Default value: #IMPLIED
This attribute is optional.

type: provides a type for a node.
Data type: CDATA
Suggested values include:
initial: initial node in a transition network
final: final node in a transition network
Default value: #IMPLIED
This attribute is optional.

adjTo: gives the IDs of the nodes which are adjacent to the current node.
Data type: IDREFS
Value: A list of IDs.
Default value: #IMPLIED
This attribute is recommended when applicable.

adjFrom: gives the IDs of the nodes which are adjacent from the current node.
Data type: IDREFS
Value: A list of IDs.
Default value: #IMPLIED
This attribute is recommended when applicable.

adj: gives the IDs of the nodes which are both adjacent to and adjacent from the current node.
Data type: IDREFS
Value: A list of IDs.
Default value: #IMPLIED
Remarks: Use this attribute instead of the *adjTo* and *adjFrom* attributes when the graph is undirected and vice versa if the graph is directed.
This attribute is recommended when applicable.

inDegree: gives the in degree of the node, the number of nodes which are adjacent from the given node.
Data type: NUMBER
Value: A nonnegative integer.
Default value: #IMPLIED
This attribute is optional.

outDegree: gives the out degree of the node, the number of nodes which are adjacent to the given node.
Data type: NUMBER
Value: A nonnegative integer.
Default value: #IMPLIED
This attribute is optional.

degree: gives the degree of the node, the number of arcs with which the node is incident.
Data type: NUMBER
Value: A nonnegative integer.
Default value: #IMPLIED
Remarks: Use this attribute instead of the *inDegree* and *outDegree* attributes when the graph is undirected and vice versa if the graph is directed.
This attribute is optional.

Example:

```
<node id=t6 label='6' inDegree=2 outDegree=0 type='final'>
```

Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: empty
May occur within: graph
May contain: EMPTY
Declarations:

```
<!ELEMENT node          - O  EMPTY                          >
<!ATTLIST node               %a.global;
          label              CDATA              #IMPLIED
          label2             CDATA              #IMPLIED
          value              IDREF              #IMPLIED
          type               CDATA              #IMPLIED
          adjTo              IDREFS             #IMPLIED
          adjFrom            IDREFS             #IMPLIED
          adj                IDREFS             #IMPLIED
```

```
            inDegree        NUMBER          #IMPLIED
            outDegree       NUMBER          #IMPLIED
            degree          NUMBER          #IMPLIED      >
```

Discussed in 23, “Graphs, Networks, and Trees,” on page 3.
This tag is recommended when applicable.

<root> (i.e. root node)

Description: represents the root node of a tree.
Attributes:

label: gives a label for a root node.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is recommended when applicable.

value: provides the value of the root, which is a feature structure or other analytic element.
Data type: IDREF
Value: A valid identifier of a feature structure or other analytic element.
Default value: #IMPLIED
This attribute is recommended when applicable.

children: provides a list of IDs of the elements which are the children of the root node.
Data type: IDREFS
Value: A list of valid identifiers.
Default value: #IMPLIED
Remarks: If the root has no children (i.e., the tree is “trivial”), then the *children* attribute must be omitted. For technical reasons, it cannot be specified as **<root children=''>**.
This attribute is required.

<!--On usage, see also remarks. -->
ord: indicates whether or not the root is ordered.
Data type: (Y | N)
Legal values are:
Y: indicates that the children of the root are ordered.
N: indicates that the children of the root are unordered.
Default value: #IMPLIED
Remarks: Use if and only if *ord=partial* is specified on the **<tree>** tag and the root has more than one child.
This attribute is recommended when applicable.

outDegree: gives the out degree of the root, the number of its children.
Data type: NUMBER
Value: A nonnegative integer.
Default value: #IMPLIED

Remarks: The in degree of the root is always 0.
This attribute is optional.

Example:

```
<root id=VP1 label=VP children='VB1 PN1' outDegree=2>
```

Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: empty
May occur within: tree
May contain: EMPTY
Declarations:

```
<!ELEMENT root          - O  EMPTY                              >
<!ATTLIST root               %a.global;
          label              CDATA              #IMPLIED
          value              IDREF              #IMPLIED
          children           IDREFS             #IMPLIED
          ord                (Y | N)            #IMPLIED
          outDegree          NUMBER             #IMPLIED        >
```

Discussed in 23.2, “Trees,” on page 15.
This tag is recommended when applicable.

Class Name: **spoken**

Description: defines additional attributes which are enabled for all elements when the base tag set for transcription of spoken texts is selected.
Attributes: [None: global and inherited attributes only.]
Part: additional tag set for spoken materials.
Member of classes:
DTD file:
Discussed in 11, “Base Tag Set for Transcriptions of Spoken Texts,” [in separate fascicle].

Class Name: **terminology** (i.e. global attributes for terminological data.)

Description: defines attributes for all elements in documents which use the base tag set for terminological data.
Attributes:

group: indicates the group (term and related elements) to which this element should be associated by specifying a string matching the *n* attribute value on an appropriate element.

Data type: CDATA
Value: any string matching the *n* attribute value on the **<term>** element to which the group is attached.
Default value: #IMPLIED
Example:

Remarks: The *group* attribute provides a specialized pointing mechanism for use within **<termEntry>** elements.
This attribute is optional.

grpPtr: indicates the group (term and related elements) to which this element should be associated by specifying its unique identifier, where this is available.
Data type: IDREF
Value: the value specified must match a value supplied as the value for an *id* attribute on some **<term>** element in the current SGML document.
Default value: #IMPLIED
Example:

Remarks: The *group* attribute provides a specialized pointing mechanism for use within **<termEntry>** elements.
This attribute is optional.

depend: indicates the parent element to which this element should be associated by specifying a string matching the *n* attribute value on an appropriate element.
Data type: CDATA
Value: any string matching the *n* attribute value on the element to which the dependent element is attached.
Default value: #IMPLIED
Example:

This attribute is optional.

depPtr: indicates the parent element to which this element should be associated by specifying its unique identifier, where this is available.
Data type: IDREF
Value: the value specified must match a value supplied as the value for an *id* attribute on some **<term>** element in the current SGML document.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

Remarks: The attributes shared by this element class are used for linking elements, possibly not adjacent in the record, which are related (e.g. a grammatical annotation and the term it describes). If no attribute is specified, the element is assumed to relate to the most recently specified **<term>** or **<otherForm>** element.
Part: base tag set for terminological data.
Member of classes:
DTD file:
Discussed in 14.2, "Tags for Terminological Data," [in separate fascicle].
Discussed in 14.3.3, "Flat Term Entries Using Group and Depend Attributes," [in separate fascicle].
Discussed in 14.4, "Overall Structure of Terminological Documents," [in separate fascicle].

<tree>

Description: encodes a tree, which is made up of a root, internal nodes, leaves, and arcs from root to leaves.
Attributes:

label: gives a label for a tree.
Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

arity: gives the maximum number of children of the root and internal nodes of the tree.
Data type: NUMBER
Value: A nonnegative integer.
Default value: #IMPLIED
This attribute is optional.

ord: indicates whether or not the tree is ordered, or if it is partially ordered.
Data type: (Y | N | partial)
Legal values are:
Y: indicates that all of the branching nodes of the tree are ordered.
partial: indicates that some of the branching nodes of the tree are ordered and some are unordered.
N: indicates that all of the branching nodes of the tree are unordered.
Default value: Y
This attribute is required.

order: gives the order of the tree, i.e., the number of its nodes.
Data type: NUMBER
Value: A nonnegative integer.
Default value: #IMPLIED

Remarks: The size of a tree is always one less than its order, hence there is no need for both a *size* and *order* attribute.
This attribute is optional.

Example:

```
<tree n='ex2' ord=partial arity=2 order=13>
   <root  label='/'  id=div1   ord=Y             children='plu1 exp1'>
   <iNode label='+'  id=plu1   ord=N parent=div1 children='exp2 exp3'>
   <iNode label='**' id=exp1   ord=Y parent=div1 children='plu2 num2.3'>
   <iNode label='**' id=exp2   ord=Y parent=plu1 children='vara1 num2.1'>
   <iNode label='**' id=exp3   ord=Y parent=plu1 children='varb1 num2.2'>
   <iNode label='+'  id=plu2   ord=N parent=exp1 children='vara2 varb2'>
   <leaf  label='a'  id=vara1        parent=exp2>
   <leaf  label='2'  id=num2.1       parent=exp2>
   <leaf  label='b'  id=varb1        parent=exp3>
   <leaf  label='2'  id=num2.2       parent=exp3>
   <leaf  label='a'  id=vara2        parent=plu2>
   <leaf  label='b'  id=varb2        parent=plu2>
   <leaf  label='2'  id=num2.3       parent=exp1>
   </tree>
```

Part: additional tag set for graphs, networks, and trees.
Member of classes: chunk
DTD file:
Data description: A root, and zero or more internal nodes and leaves, but if there is an internal node, there must also be at least one leaf.
May occur within: forest
May contain: iNode leaf root
Declarations:

```
<!ELEMENT tree          - -  ((leaf | iNode)*, root, (leaf |
                             iNode)*)                            >
<!ATTLIST tree               %a.global;
          label              CDATA              #IMPLIED
          arity              NUMBER             #IMPLIED
          ord                (Y | N | partial)   Y
          order              NUMBER             #IMPLIED         >
```

Discussed in 23.2, “Trees,” on page 15.
This tag is recommended when applicable.

<triangle>

(i.e. Underspecified embedding tree, so called because of its) characteristic shape when drawn.
Description: provides for an underspecified **<eTree>**, that is, an **<eTree>** with information left out.
Attributes:

label: gives a label for an underspecified embedding tree.

Data type: CDATA
Value: A character string.
Default value: #IMPLIED
This attribute is optional.

value: provides the value of a triangle, which is the SGML identifier of a feature structure or other analytic element.
Data type: IDREF
Value: A valid identifier of a feature structure or other analytic element.
Default value: #IMPLIED
This attribute is recommended when applicable.

Example:

```
<triangle label='NP'><eLeaf label='the periscope'></triangle>
```

Part: additional tag set for graphs, networks, and trees.
Member of classes:
DTD file:
Data description: An optional label or feature structure followed by zero or more embedding trees, triangles, or embedding leafs.
May occur within: eTree forest triangle
May contain: eLeaf eTree triangle
Declarations:

```
<!ELEMENT triangle      - -  ((eTree | triangle | eLeaf)*)      >
<!ATTLIST triangle           %a.global;
          label              CDATA              #IMPLIED
          value              IDREF              #IMPLIED        >
```

Discussed in 23.3, “Another Tree Notation,” on page 19.
This tag is optional.

Usability Report and Comment

If you have comments of any kind on the current draft of the TEI Guidelines, please complete this form and return it to one of the following addresses:

C. M. Sperberg-McQueen, Computer Center (M/C 135), University of Illinois at Chicago, 1940 W. Taylor Room 124, Chicago IL 60612-7352, USA. Bitnet: U35395@UICVM, Internet: U35395@uicvm.uic.edu

Lou Burnard, Oxford University Computing Service, 13 Banbury Road, Oxford OX2 6NN, UK. Fax: +44 (865) 273275, Janet: LOU@UK.AC.OXFORD.VAX Internet: LOU@vax.oxford.ac.uk

Further copies of this form may be downloaded from LISTSERV @ UICVM (or listserv @ uicvm.uic.edu). Send a mail message containing the line 'GET TEIURC MEMO TEI-L'

Your name:
Your postal address:

Your e-mail address (if any):
Your occupation:
Your academic background:
Your immediate reactions to the Guidelines (please tick or cross one only):

Relevance to your present concerns or interests: high medium low
Importance to your future research plans: high medium low
Comprehensibility/usability of current draft: high medium low

Detailed technical comment: [Please feel free to comment on any part of the Guidelines or in general. Comments relating to specific parts of the text are most helpful if they include a section or chapter number]

Thank you!

Temporary Index