21 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.

- 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 21.2 *Trees* and 21.3 *Another Tree Notation*.[151]
- *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 22 *Tables, Formulae, and Graphics*.

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

```
<!-- 21.: Graphs, networks and trees-->
<!--declarations from 21.1: Graphs inserted here -->
<!--declarations from 21.2: Trees (basic method) inserted here -->
<!--declarations from 21.3: Trees (alternate method) inserted here -->
<!-- end of 21.-->
```

This tag set is made available as described in 3.3 *Invocation of the TEI DTD*; in a document which uses the markup described in this chapter, the document type declaration should contain the following declaration for the entity TEI.nets:

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

The entire document type declaration for an XML document using this additional tag set together with the prose base might look like this:

```
<!DOCTYPE TEI.2 PUBLIC "-//TEI P4//DTD Main Document Type//EN"
                       "http://www.tei-c.org/P4X/DTD/tei2.dtd" [
    <!ENTITY % TEI.XML   'INCLUDE' >
    <!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.

21.1 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:

[151] 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).

type describes the type of graph.
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

label gives a label for a graph.
Values A character string.

order states the order of the graph, i.e., the number of its nodes.
Values A positive integer.

size states the size of the graph, i.e., the number of its arcs.
Values A non-negative integer.

<node> encodes a node, a possibly labeled point in a graph. Attributes include:

label gives a label for a node.
Values A character string.

label2 gives a second label for a node.
Values A character string.

value provides the value of a node, which is a feature structure or other analytic element.
Values A valid identifier.

type provides a type for a node.
Suggested values include:
`initial` initial node in a transition network
`final` final node in a transition network

adjFrom gives the identifiers of the nodes which are adjacent from the current node.
Values A list of identifiers.

adjTo gives the identifiers of the nodes which are adjacent to the current node.
Values A list of identifiers.

adj gives the identifiers of the nodes which are both adjacent to and adjacent from the current node.
Values A list of identifiers.

inDegree gives the in degree of the node, the number of nodes which are adjacent from the given node.
Values A non-negative integer.

outDegree gives the out degree of the node, the number of nodes which are adjacent to the given node.
Values A non-negative integer.

degree gives the degree of the node, the number of arcs with which the node is incident.
Values A non-negative integer.

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

label gives a label for an arc.
Values A character string.

label2 gives a second label for an arc.
Values A character string.

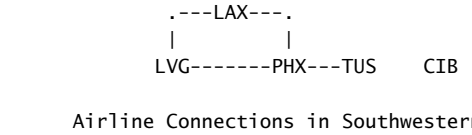
from gives the identifier of the node which is adjacent from this arc.
Values The identifier of a node.

to gives the identifier of the node which is adjacent to this arc.
Values The identifier of a node.

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 annotated 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.



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'
       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 label attribute on the <graph> element records a label for the graph; 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 graph.

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'
       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>
```

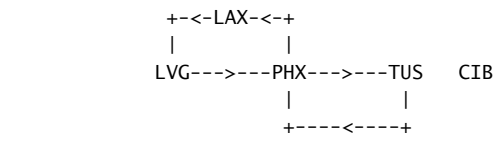
Note that each arc is represented twice in this encoding of the graph. For example, the existence of the arc from LAX to LVG can be inferred from each of the first two <node> elements in the graph. This

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'
       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 arrowheads in the following diagram.



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'
       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'
       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.

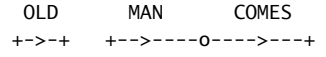
```
<graph type='directed'
       id='RDG1'
       label='Selected Airline Routes in Southwestern USA'
       order='5'
       size='5'>
   <node label='LAX' id='LAX'/>
   <node label='LVG' id='LVG'/>
   <node label='PHX' id='PHX'/>
   <node label='TUS' id='TUS'/>
   <node label='CIB' id='CIB'/>
   <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>
```

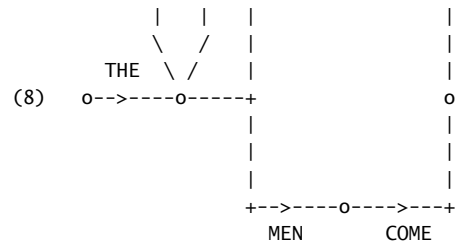
The formal declarations of the <graph>, <node> and <arc> elements are as follows.

```
<!-- 21.1: Graphs-->
<!ELEMENT graph %om.RR; (  (( node, (%m.Incl;)* )+, ( arc,  (%m.Incl;)* )*)
                  | (( arc,  (%m.Incl;)* )+, ( node, (%m.Incl;)* )+)) >
<!ATTLIST graph
      %a.global;
      type CDATA #IMPLIED
      label CDATA #IMPLIED
      order CDATA #IMPLIED
      size CDATA #IMPLIED
      TEIform CDATA 'graph'  >
<!ELEMENT node %om.RO; 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 CDATA #IMPLIED
      outDegree CDATA #IMPLIED
      degree CDATA #IMPLIED
      TEIform CDATA 'node'  >
<!ELEMENT arc %om.RO; EMPTY>
<!ATTLIST arc
      %a.global;
      label CDATA #IMPLIED
      label2 CDATA #IMPLIED
      from IDREF #REQUIRED
      to IDREF #REQUIRED
      TEIform CDATA 'arc'  >
<!-- end of 21.1-->
```

21.1.1 Transition Networks

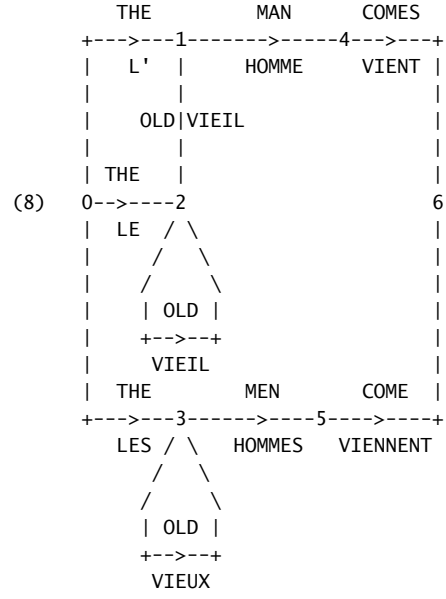
For encoding transition networks and other kinds of directed graphs in which distinctions among types of nodes must be made, the type attribute is provided for <node> elements. In the following example, the *initial* and *final* nodes (or *states*) of the network are distinguished. It can be understood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






```
<graph type='transition network'
       id='SS8'
       label='(8)'
       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 state 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 annotated with numbers, for convenience.

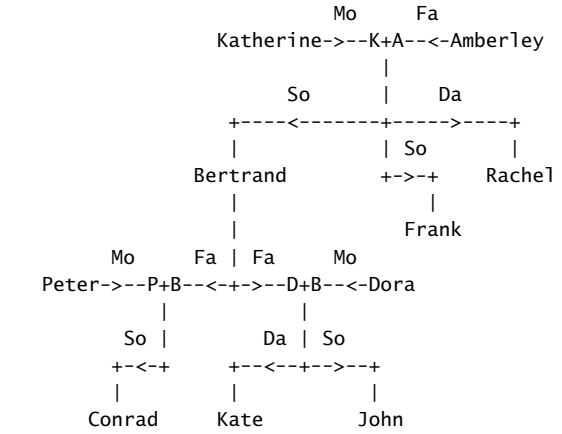



```
<graph type='transducer'
       order='7'
       size='10'>
   <node id='T0' label='0' inDegree='0' outDegree='3' type='initial'/>
   <node id='T1' label='1' inDegree='2' outDegree='1'/>
   <node id='T2' label='2' inDegree='2' outDegree='2'/>
   <node id='T3' label='3' inDegree='2' outDegree='2'/>
   <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='VIENNENT'/>
</graph>
```

21.1.2 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 16.3 *Feature, Feature-Structure and Feature-Value Libraries*). We can use the value attribute on <node> elements to point to those feature structures. Assume that, in some particular representation of the graph, nodes representing females are framed by circles, nodes representing males are framed by boxes, and nodes representing parents are framed by diamonds.




```
<graph type='family tree' order='13' size='12'>
  <node id='KATHR'  label='Katherine'
        value='kr1' inDegree='0' outDegree='1'/>
  <node id='AMBER'  label='Amberley'
        value='ar1' inDegree='0' outDegree='1'/>
  <node id='KAR'  label='K+A'
        inDegree='2' outDegree='3'/>
  <node id='BERTR'  label='Bertrand'
        value='br1' inDegree='1' outDegree='2'/>
  <node id='PETER'  label='Peter'
        value='pr1' inDegree='0' outDegree='1'/>
  <node id='DORAR'  label='Dora'
        value='dr1' inDegree='0' outDegree='1'/>
  <node id='PBR'  label='P+B'
        inDegree='2' outDegree='1'/>
  <node id='DBR'  label='D+B'
        inDegree='2' outDegree='2'/>
  <node id='FRANR'  label='Frank'
        value='fr1' inDegree='1' outDegree='0'/>
  <node id='RACHR'  label='Rachel'
        value='rr1' inDegree='1' outDegree='0'/>
  <node id='CONRR'  label='Conrad'
        value='cr1' inDegree='1' outDegree='0'/>
  <node id='KATER'  label='Kate'
        value='kr2' inDegree='1' outDegree='0'/>
  <node id='JOHNR'  label='John'
        value='jr1' inDegree='1' outDegree='0'/>
```

```
<arc label='Mo' from='KathR' to='KAR'/>
<arc label='Fa' from='AmbeR' to='KAR'/>
<arc label='So' from='KAR' to='BertR'/>
<arc label='So' from='KAR' to='FranR'/>
<arc label='Da' from='KAR' to='RachR'/>
<arc label='Mo' from='PeteR' to='PBR'/>
<arc label='Fa' from='BertR' to='PBR'/>
<arc label='So' from='PBR' to='ConrR'/>
<arc label='Mo' from='DoraR' to='DBR'/>
<arc label='Fa' from='BertR' to='DBR'/>
<arc label='Da' from='DBR' to='KateR'/>
<arc label='So' from='DBR' to='JohnR'/>
</graph>
```

21.1.3 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 sherrifdom 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:

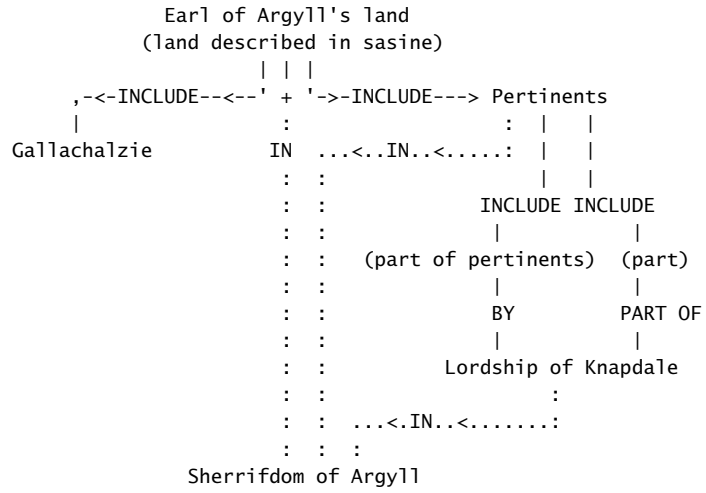
- containment (INCLUDE)
- location within (IN)
- contiguity (BY)
- constituency (PART OF)

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 15 *Simple Analytic Mechanisms*.[152]

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.



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 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>
```

[152] 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 16 *Feature Structures*.

21.2 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.

Values A nonnegative integer.

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.

Values A nonnegative integer.

<root> represents the root node of a tree. Attributes include:

label gives a label for a root node.

Values A character string.

value provides the value of the root, which is a feature structure or other analytic element.

Values A valid identifier of a feature structure or other analytic element.

children provides a list of identifiers of the elements which are the children of the root node.

Values A list of valid identifiers.

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.

Values A nonnegative integer.

<iNode> represents an intermediate (or internal) node of a tree. Attributes include:

label gives a label for an intermediate node.

Values A character string.

value provides the value of an intermediate node, which is a feature structure or other analytic element.

Values A valid identifier of a feature structure or other analytic element.

children provides a list of identifiers of the elements which are the children of the intermediate node.

Values A list of identifiers.

parent provides the identifier of the element which is the parent of this node.

Values The identifier of the parent 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 identifier of the element which this node follows.

Values The identifier of another intermediate node or leaf of the tree.

outDegree gives the out degree of an intermediate node, the number of its children.

Values A nonnegative integer.

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

label gives a label for a leaf.

Values A character string.

value provides the value of a leaf, which is a feature structure or other analytic element.

Values A valid identifier of a feature structure or other analytic element.

parent provides the identifier of parent of a leaf.

Values The identifier of the parent node.

follow provides an identifier of an element which this leaf follows.

Values The identifier of another intermediate node or leaf of the tree.

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 attribute ord="N" on those two <iNode> elements, the attribute

ord="Y" on the <root> and other <iNode> elements, and the attribute 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 xxx symbol indicates that the arc from VB to PT crosses the arc from VP to PN.

```
       ,--VP--,
       |      |
   ,--VB-----xxx--,
   |           |   |
   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.

```
<!-- 21.2: Trees (basic method)-->
<!ELEMENT tree %om.RR; ((leaf | iNode)*, root, (leaf | iNode)*)>
<!ATTLIST tree
      %a.global;
      label CDATA #IMPLIED
      arity CDATA #IMPLIED
      ord (Y | N | partial) "Y"
      order CDATA #IMPLIED
      TEIform CDATA 'tree'  >
<!ELEMENT root %om.RO; EMPTY>
<!ATTLIST root
      %a.global;
      label CDATA #IMPLIED
      value IDREF #IMPLIED
      children IDREFS #IMPLIED
      ord (Y | N) #IMPLIED
      outDegree CDATA #IMPLIED
      TEIform CDATA 'root'  >
<!ELEMENT iNode %om.RO; 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 CDATA #IMPLIED
      TEIform CDATA 'iNode'  >
<!ELEMENT leaf %om.RO; EMPTY>
<!ATTLIST leaf
      %a.global;
      label CDATA #IMPLIED
      value IDREF #IMPLIED
      parent IDREF #IMPLIED
      follow IDREF #IMPLIED
      TEIform CDATA 'leaf'  >
<!-- end of 21.2-->
```

21.3 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 21.2 *Trees*, 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.

Values A character string.

value provides the value of an embedding tree, which is a feature structure or other analytic element.

Values A valid identifier of 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.

Values A character string.

value provides the value of a triangle, which is the identifier of a feature structure or other analytic element.

Values A valid 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.

Values A character string.

value provides the value of an embedding leaf, which is a feature structure or other analytic element.

Values A valid identifier of a feature structure or other analytic element.

<forest> provides for groups of rooted trees. Attributes include:

type identifies the type of the forest.

Values A character string.

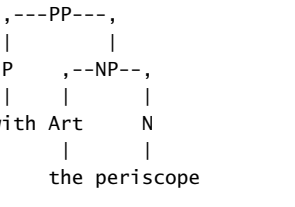
<forestGrp> provides for groups of forests. Attributes include:

type identifies the type of the forest group.

Values A character string.

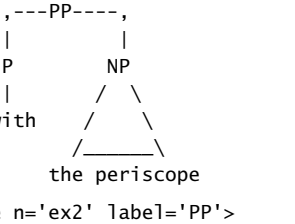
Like the <root>, <iNode> and <leaf> of a <tree>, the <eTree>, <triangle> and <eLeaf> elements may also have label and value attributes.

To illustrate the use of the <eTree> and <eLeaf> elements, here is an encoding of the second example in section 21.2 *Trees*, repeated here for convenience.



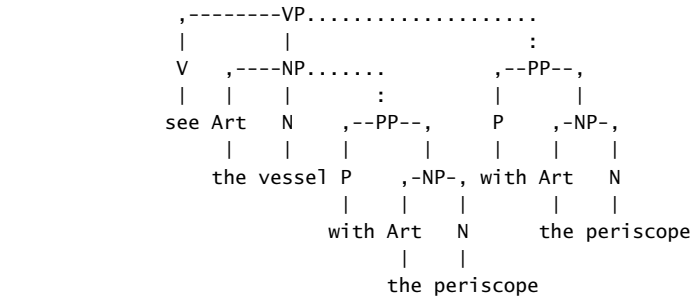

```
<eTree n='ex1' label='PP'>
   <eTree label='P'><eLeaf label='with'/></eTree>
   <eTree label='NP'>
      <eTree label='Art'><eLeaf label='the'/></eTree>
      <eTree label='N'><eLeaf label='periscope'/></eTree>
      </eTree>
   </eTree>
```

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




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

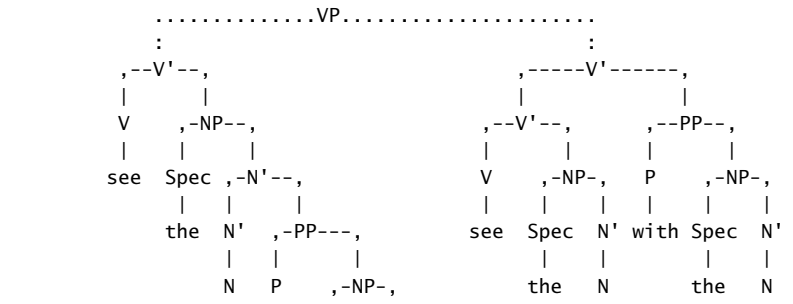
Ambiguity involving alternative tree structures associated with the same terminal sequence can be encoded relatively conveniently using a combination of the exclude and copyOf attributes described in sections 14.8 *Alternation* and 14.6 *Identical Elements and Virtual Copies*. In the simplest case, an <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.




```
<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 14.8 *Alternation*.

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 Jackendoff,[153] 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.



[153] R. Jackendoff, *X-Bar Syntax*, 1977

```
          |   |    |    |                |            |
      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='VBARB' exclude='VBARA' 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'.[154]

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

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

```
      </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 14.4 *Correspondence and Alignment*) 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.

```
<!-- 21.3: Trees (alternate method)-->
<!ELEMENT eTree %om.RR; ((eTree |  triangle | eLeaf )*)>
<!ATTLIST eTree
      %a.global;
      label CDATA #IMPLIED
      value IDREF #IMPLIED
      TEIform CDATA 'eTree'  >
<!ELEMENT triangle %om.RR; ((eTree | triangle | eLeaf)*)>
<!ATTLIST triangle
      %a.global;
      label CDATA #IMPLIED
      value IDREF #IMPLIED
      TEIform CDATA 'triangle'  >
<!ELEMENT eLeaf %om.RO; EMPTY>
<!ATTLIST eLeaf
      %a.global;
      label CDATA #IMPLIED
      value IDREF #IMPLIED
      TEIform CDATA 'eLeaf'  >
<!ELEMENT forest %om.RR; ((tree | eTree | triangle)+)>
<!ATTLIST forest
      %a.global;
      type CDATA #IMPLIED
      TEIform CDATA 'forest'  >
<!ELEMENT forestGrp %om.RR; ((forest)+)>
<!ATTLIST forestGrp
      %a.global;
      type CDATA #IMPLIED
      TEIform CDATA 'forestGrp'  >
<!-- end of 21.3-->
```