

# Corpus Linguistics (L615) Linguistic Annotation

Markus Dickinson

Department of Linguistics, Indiana University  
Spring 2009

---

Corpus Linguistics

Linguistic  
Annotation

Corpus markup

Mark-up schemes

Character encodings

Corpus annotation

Motivation

How to annotate

Levels of Annotation

Text Processing

Tokenization

Lemmatization

Standalone  
annotation

More on XML

## Corpus markup

Mark-up schemes  
Character encodings

## Corpus annotation

Motivation  
How to annotate  
Levels of Annotation

## Text Processing

Tokenization  
Lemmatization

## Standalone annotation

More on XML

Corpus **mark-up** is information about the corpus included in the corpus itself

- ▶ some of it governs formatting, printing, other processing

Mark-up is important to:

- ▶ help recover the original context
- ▶ allow for a wider range of questions to be explored (e.g., sociolinguistic variables)
- ▶ encode extralinguistic details (e.g., laughs, formulas)

We will use the term **annotation** to refer specifically to linguistic information encoded in the text

- ▶ Generally speaking, annotation is a form of markup
- ▶ Non-linguistic information also encoded in markup (meta-data)
  - ▶ i.e. information about the text: author, version, date, information about author, encoding

## Corpus markup

Mark-up schemes  
Character encodings

## Corpus annotation

Motivation  
How to annotate  
Levels of Annotation

## Text Processing

Tokenization  
Lemmatization

## Standalone annotation

## More on XML

Meta-data systematically needs to be kept separate from the corpus data

A general idea is to use **attributes** and corresponding **values**

- ▶ e.g., attribute `author` has a value `William Shakespeare`
- ▶ We'll see this encoded in XML in ways such as:
  - ▶ `<... author="William Shakespeare" ...>`
  - ▶ `<author name="William Shakespeare">`

# Text Encoding Standards

- ▶ TEI: Text Encoding Initiative
- ▶ CES: Corpus Encoding Standard
- ▶ XCES: XML version of CES

If you annotate a corpus, make it conform to XCES as much as possible

# Text Encoding Initiative (TEI)

TEI-compliant documents are broken into 2 parts:

1. Header = information about the document
  - ▶ File description (<fileDesc>): bibliographic information
  - ▶ Encoding description (<encodingDesc>): relationship to the source
  - ▶ Text profile (<profileDesc>): non-bibliographic information (e.g., setting of the text, languages used, etc.)
  - ▶ Revision history (<revisionDesc>): changes made
2. Body = the original text itself

TEI documents can be encoded in different mark-up languages, e.g., SGML and XML

Mark-up languages generally have the following aspects

- ▶ Tags, which come in both start (e.g., <s>) and end (e.g., </s>) forms
- ▶ Attributes and values, encoded within the start tag (e.g., <s num="1">)

```
<w POS=AT0>the</w>
```

XML stands for: eXtensible Markup Language

- ▶ in contrast to HTML, XML does not have built-in “meaning” for labels: you must define your own tags!

An XML document consists of two parts: DTD (or XML Schema) and the real document

- ▶ DTD (Document Type Definition) defines structure of document
  - ▶ XML Schema does the same thing, but in a different format
- ▶ XML documents can be validated, i.e., checked against rules in DTD

We'll save a more thorough discussion of XML for the end of this unit

# CES: Corpus Encoding Standard

TEI is for any text document and, as such, defines approximately 450 elements

- ▶ CES simplifies this system in order to encode language corpora

CES covers:

- ▶ document-wide mark-up
- ▶ gross structural mark-up, for structural units of text
- ▶ mark-up for sub-paragraph structures

CES documents can be validated at metalanguage, syntactic, or semantic levels

# Character encodings

Corpus has to specify character encoding of the text, as different encodings exist (ASCII, ISO-8859-1, etc.)

Unicode has a single representation for every possible character

- ▶ Version 3.2 has codes for 95,221 characters from alphabets, syllabaries and logographic systems
- ▶ Unicode has three versions
  - ▶ UTF-32 (32 bits): direct representation
  - ▶ UTF-16 (16 bits):  $2^{16} = 65536$
  - ▶ UTF-8 (8 bits):  $2^8 = 256$

Unicode (especially UTF-8) is the standard for XML documents

Corpus annotation = “interpretative, linguistic information”  
added to a corpus

- ▶ Corpus mark-up is relatively objective factual information
- ▶ Corpus annotation is more subjective, interpretative information

## Why would we want annotation?

- ▶ for training NLP tools
- ▶ for finding examples
  - ▶ what is the plural form of `fish`?
  - ▶ which nouns can occur as bare nouns, without a determiner?
  - ▶ are there subjectless sentences in German?
    - Yes, e.g. `Mir ist kalt.` (To me is cold.)
  - ▶ is it possible in English to have something between a noun and its modifying relative clause?

Annotation makes it possible to find phenomena that would otherwise disappear in masses of data

# Potential benefits of annotation

Corpus annotation adds much value to a corpus

- ▶ Extracting information is easier:
  - ▶ Can easily isolate *left* in its adjective uses
  - ▶ Can find information on a language you don't speak
- ▶ Corpus is more reusable
  - ▶ Insights are more accessible to others
- ▶ Corpus is more multifunctional
- ▶ Annotation provides a clear record of analysis, open to future scrutiny
  - ▶ Decisions are more objective, reproducible

# Potential criticisms of annotation

- ▶ Annotation is cluttered
  - ▶ But: it can be easily ignored
- ▶ Annotation imposes one particular linguistic analysis
  - ▶ But: users can reject the analysis, and annotation at least makes the analysis process clearer
- ▶ Annotation makes a corpus less accessible and expandable
  - ▶ But: corpora can be extended without annotation, if need be
- ▶ Annotation cannot be done completely consistently
  - ▶ But: humans are fallible, and there are ways to ensure better consistency

# Leech's Seven Maxims of Annotation

1. It should be possible to remove the annotation from an annotated corpus in order to revert to the raw corpus.
2. It should be possible to extract the annotations by themselves from the text. This is the flip side of maxim 1.
  - ▶ Taking points 1. and 2. together, the annotated corpus should allow the maximum flexibility for manipulation by the user.
3. The annotation scheme should be based on guidelines which are available to the end user.
4. It should be made clear how and by whom the annotation was carried out.

# Leech's Seven Maxims (2)

5. The end user should be made aware that the corpus annotation is not infallible, but simply a potentially useful tool.
6. Annotation schemes should be based as far as possible on widely agreed and theory-neutral principles.
7. No annotation scheme has the a priori right to be considered as a standard. Standards emerge through practical consensus.

# How corpus annotation is achieved

Some options:

- ▶ Fully manually
  - ▶ Pro: has the potential to be of highest quality
  - ▶ Con: time-consuming + humans are prone to errors
- ▶ Fully automatically (assuming appropriate technology)
  - ▶ Pro: quick and consistent
  - ▶ Con: will often be consistently wrong
- ▶ Semi-automatically, e.g., automatic analysis + manual post-editing
  - ▶ Pro: Can combine the best of both worlds
  - ▶ Con: Have to avoid the pitfalls of each

# Levels of linguistic annotation

- ▶ morphological annotation (e.g. inflection, derivation, compounding)
- ▶ morpho-syntactic annotation: part-of-speech (POS) tagging
- ▶ syntactic annotation (e.g. named entities, phrasal chunking, full syntactic analysis)
- ▶ semantic annotation (e.g. word-sense disambiguation, anaphora and coreference resolution, information structure)
- ▶ discourse annotation (e.g. dialog turns, speech acts)

Corpus Linguistics

Linguistic  
Annotation

Corpus markup

Mark-up schemes

Character encodings

Corpus annotation

Motivation

How to annotate

Levels of Annotation

Text Processing

Tokenization

Lemmatization

Standalone  
annotation

More on XML

# Step by Step Annotation

- ▶ tokenization
- ▶ lemmatization / morphological analysis
- ▶ part-of-speech tagging
- ▶ named-entity recognition
- ▶ partial parsing
- ▶ full syntactic parsing
- ▶ semantic and discourse processing

See section A4.4 of the book for a more thorough analysis of each of these

# Annotation uses

- ▶ POS annotation: compare occurrences of word classes
- ▶ Lemmatization: study distribution of lexemes for lexicography
- ▶ Parsing: study clauses types in a language or teach grammatical analysis
- ▶ Semantic annotation: analyze content

Other types of annotation can cover specific purposes: stylistic annotation, error tagging, problem-oriented annotation

We'll examine POS and syntactic annotation in greater detail later in the semester

# Preprocessing the Text: Tokenization

Tokenization refers to the annotation step of dividing the input text into units called *tokens*.

Each tokens consists of one of the following:

- ▶ a morpho-syntactic word
- ▶ a punctuation mark or a special character (e.g. &, @, %)
- ▶ a number

# Tokenization – Example

before tokenization:

Milton wrote "Paradise Lost." Then his wife dies  
and he wrote "Paradise Regained."

after tokenization:

Milton wrote " Paradise Lost . " Then his wife  
dies and he wrote " Paradise Regained . "

Corpus Linguistics

Linguistic  
Annotation

Corpus markup

Mark-up schemes

Character encodings

Corpus annotation

Motivation

How to annotate

Levels of Annotation

Text Processing

Tokenization

Lemmatization

Standalone  
annotation

More on XML

# Why is Tokenization Non-Trivial?

- ▶ disambiguation of punctuation  
e.g. period can occur inside cardinal numbers, after ordinals, after abbreviations, at end of sentences
- ▶ recognition of complex words
  - ▶ compounds, e.g. bank transfer fee, US-company
  - ▶ mergers, e.g. don't, England's, French: t'aime
  - ▶ multiwords, e.g. complex prepositions provided that, in spite of

Languages such as Chinese present more challenging segmentation issues

# Lemmatization

- ▶ refers to the process of relating individual word forms to their citation form (lemma) by means of morphological analysis  
e.g. stopped  $\Rightarrow$  stop
- ▶ provides a means to distinguish between the total number of word tokens and distinct lemmata that occur in a corpus  
e.g. helps to find all occurrences of buy
- ▶ is indispensable for highly inflectional languages which have a large number of distinct word forms for a given lemma

# Lemmatization – German Example

wie	wie	+Adv+Wh+#lex+COWIE
wie	wie	+Conj+Coord+#lex+COWIE
wie	wie	+Conj+Subord+#lex+COWIE
sie	sie	+Pron+Pers+3P+Pl+Fem+Nom+#lex+PERSPRO
sie	sie	+Pron+Pers+3P+Sg+Fem+Nom+#lex+PERSPRO
offenbar	offenbaren	+Verb+Imp+2P+Sg+#lex+VVFIN
offenbar	offenbar	+Adj+Pos+Pred+#lex+ADJD
gedacht	gedenken	+Verb+PPast+#lex+VVPP
gedacht	dachen	+Verb+PPast+#lex+VVPP
gedacht	denken	+Verb+PPast+#lex+VVPP
hat	haben	+Verb+Indc+3P+Sg+Pres+#lex+VAFIN

- ▶ **XEROX Morphological Analyzer:** comprehensive morphological analyzers for many languages including English, French, Dutch, German, Hungarian, Italian, Portuguese, Czech, Danish, Finnish, Norwegian, Polish, Russian, Turkish.  
link: <http://www.xrce.xerox.com/competencies/content-analysis/toolhome.en.html>
- ▶ **Lingsoft:** morphological analyzers for English, Danish, German, Swedish, and Finnish  
link: <http://www.lingsoft.fi/demos.html>

# Morphological Analysis

Xerox:

half half+Adj

half half+Adv

half half+Noun+Sg

Lingsoft:

"<half>"

"half" <Quant> DET PRE SG/PL @QN>

"half" <NonMod> <Quant> PRON SG/PL

"half" N NOM SG

"half" ADV

Corpus Linguistics

Linguistic  
Annotation

Corpus markup

Mark-up schemes

Character encodings

Corpus annotation

Motivation

How to annotate

Levels of Annotation

Text Processing

Tokenization

Lemmatization

Standalone  
annotation

More on XML

# Standalone annotation

Instead of *embedding* annotation in the base document, one can use standalone, or standoff, annotation:

- ▶ The annotation is linked to particular points in the original document

Advantages:

- ▶ Allows base documents to be treated as read-only & can be distributed separately
- ▶ Allows for overlapping annotation and alternative annotation
- ▶ Allows for new annotation levels to be easily added

Disadvantages:

- ▶ Potentially difficult to allow for annotation to point to other annotation
- ▶ Less pre-built tools for standoff annotation

# Example XML document

We'll look more closely at XML

- ▶ Some of this material is adapted from:  
<http://www.w3schools.com/xml/>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

# XML tags

In the previous example, there are several things to note about the tags, i.e., the things in angled brackets (<...>):

- ▶ There has to be one root tag—in this case, <note> is the root tag
- ▶ Every (opening) tag needs a closing tag:  
<heading>Reminder</heading> is legitimate;  
<heading>Reminder is not
- ▶ Along with that, tags must be properly nested:  
<b><i>word</i></b> is legitimate;  
<b><i>word</b></i> is not

So, you'll wind up with a structure like:

```
<root>  
  <child>  
    <subchild>.....</subchild>  
  </child>  
</root>
```

Each tag (or element) can have a variety of attributes  
(provided such attributes have been declared; see below)

- ▶ Attributes are noted within an element tag; there can be multiple attributes
- ▶ Attribute values are put in quotes after the attribute
- ▶ Some examples:

```
<note date="2/23/2006">
```

```
<note date="2/23/2006" author="joe shmoe">
```

- ▶ This last one is equivalent to:

```
<note author="joe shmoe" date="2/23/2006">
```

# Comments and whitespace

Comments are added in between `<!--` and `-->`

Does whitespace matter? ... Yes and no.

- ▶ Within tags, each whitespace character is treated as significant (unlike, e.g., HTML)
- ▶ But it doesn't matter how much whitespace you put between different tags, or if you indent a child tag
  - ▶ The data is structured in and of itself, regardless of whitespace.

# Attributes vs. Elements (1)

There is often a question of where to store information: in element text, in attributes, as children elements, ... Here are three different ways to say the “same” thing

```
<note date="12/11/2002">  
<to>Tove</to>  
<from>Jani</from>  
<heading>Reminder</heading>  
<body>Don't forget me this weekend!</body>  
</note>
```

# Attributes vs. Elements (2)

```
<note>
<date>12/11/2002</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Corpus Linguistics

Linguistic  
Annotation

Corpus markup

Mark-up schemes

Character encodings

Corpus annotation

Motivation

How to annotate

Levels of Annotation

Text Processing

Tokenization

Lemmatization

Standalone  
annotation

More on XML

## Attributes vs. Elements (3)

```
<note>
<date>
  <day>12</day>
  <month>11</month>
  <year>2002</year>
</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

If you actually want to do something with the day, month, or year, this last example is the most effective (i.e., most structured)

In general, putting data in attributes needs to be done with care:

- ▶ attributes cannot contain multiple values (child elements can)
- ▶ attributes are not easily expandable (for future changes)
- ▶ attributes cannot describe structures (child elements can)
- ▶ attributes are more difficult to manipulate by program code
- ▶ attribute values are not as easy to test against a Document Type Definition (DTD) - which is used to define the legal elements of an XML document

However, converting all attributes to elements can result in an inflated structure

# Linguistic example

So, what do we do with corpora? Well, this won't typically be your problem, as someone else will have decided, but here are some possibilities:

```
<terminal tag="NN">dog</terminal>
```

```
<terminal word="dog" tag="NN"></terminal>
```

```
<!-- or: <terminal word="dog" tag="NN"/> -->
```

```
<terminal>  
  <word>dog</word>  
  <tag>NN</tag>  
</terminal>
```

We want to know what makes up a valid XML file, i.e., we want to make sure our corpus is annotated in a consistent fashion; if not, the DTD allows us to automatically find out

- ▶ We can specify the Document Type Definition (DTD) in a separate file or at the top of the XML file
- ▶ It tells us what the valid elements are; what children those elements can have; what attributes; etc.

# Document Type Definition (DTD) example

```
<!DOCTYPE note [  
  <!ELEMENT note (to,from,heading,body)>  
  <!ELEMENT to      (#PCDATA)>  
  <!ELEMENT from    (#PCDATA)>  
  <!ELEMENT heading (#PCDATA)>  
  <!ELEMENT body    (#PCDATA)>  
>
```

- ▶ Document is of type note (i.e. that's the root element)
- ▶ The element note has four different children
- ▶ The other four elements simply have parsed character data (#PCDATA) as their content

# More complex definitions

- ▶ Declaring one or more of the same element:  
`<!ELEMENT nonterminal (terminal+)>`
- ▶ Declaring zero or more of the same element:  
`<!ELEMENT nonterminal (terminal*)>`
- ▶ Declaring mixed content, in any order, any number of times:  
`<!ELEMENT terminal (word|tag|#PCDATA)*>`

Declaring attributes are slightly more complicated

DTD example:

```
<!ATTLIST payment type CDATA "check">  
<!-- element-name att-name valid-content default-value -->
```

XML example:

```
<payment type="check" />
```

You can set attribute values to be **#IMPLIED** (not necessary), **#REQUIRED**, or **#FIXED**

# XML Schema (XSD)

DTDs are in the process of being replaced by XML Schemas (XSDs)

- ▶ See <http://www.w3schools.com/schema/>

Why XSDs?

- ▶ More support for data types, including, e.g., being easier to convert between different data types
- ▶ XSDs use XML syntax, so you don't have to relearn a new syntax
- ▶ Easier to validate correctness

Our earlier DTD would have the equivalent XSD on the next page ...

# XSD example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://..." targetNamespace="http://..."
  xmlns="http://..." elementFormDefault="qualified">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# XML for linguistic purposes

- ▶ XML allows us to add structured information to a text
  - ▶ By adding XML annotation, we are able to preserve the original document—i.e., we only add to it
- ▶ The complexity of the XML used depends on exactly what the task is.
- ▶ Using XML and a corresponding DTD also allows corpus designers or software designers to specify how the annotation information should be formatted

## *New York Times* data from the English Gigaword corpus:

```
<DOC id="NYT19940701.0001" type="story" >
<HEADLINE>
WITNESS SAYS O.J. SIMPSON BOUGHT KNIFE WEEKS BEFORE SLAYINGS
</HEADLINE>
<DATELINE>
LOS ANGELES (BC-SIMPSON-KILLINGS-1stLd-3Takes-Writethru-LADN)
</DATELINE>
<TEXT>
<P>
With the nation's attention riveted again on a Los
Angeles courtroom, a knife dealer testified that O.J. Simpson
bought a 15-inch knife five weeks before the slashing deaths of his
ex-wife and her friend.
</P>
...
<P>
'She frequented the restaurant quite often,' DeBello said.
</P>
<P>
(STORY CAN END HERE. OPTIONAL 2ND TAKE FOLLOWS.)
</P>
</TEXT>
</DOC>
```

Corpus markup

Mark-up schemes

Character encodings

Corpus annotation

Motivation

How to annotate

Levels of Annotation

Text Processing

Tokenization

Lemmatization

Standalone  
annotation

More on XML

# Part of Gigaword DTD

```
<!ELEMENT DOC      - - (HEADLINE*,
                        DATELINE*,
                        TEXT*)      >

<!-- fields of "DOC" -->
<!ELEMENT HEADLINE - - (#PCDATA)  >
<!ELEMENT DATELINE - - (#PCDATA)  >
<!ELEMENT TEXT     - - (P* | #PCDATA)+ >

<!-- fields of "TEXT" -->
<!ELEMENT P        - - (#PCDATA)  >

<!--Entities -->
<!ENTITY amp      "&" >
<!ENTITY AMP      "&" >
<!ENTITY #DEFAULT SYSTEM >

<!ATTLIST DOC    id    CDATA #REQUIRED
                  type  CDATA #REQUIRED >

]>
```

# Adding POS information: BNC

```
</c></s><s n="45" p="n" teiform="s">  
<w type="av0" teiform="w">Commonly </w>  
<w type="vvn-vvd" teiform="w">held </w>  
<w type="nn2" teiform="w">ideas </w>  
<w type="vrb" teiform="w">restrict </w>  
<w type="at0" teiform="w">the </w>  
<w type="aj0" teiform="w">social </w>  
<w type="nn1" teiform="w">role </w>  
<w type="cjc" teiform="w">and </w>  
<w type="nn1" teiform="w">status </w>  
<w type="prf" teiform="w">of </w>  
<w type="ajc" teiform="w">older </w>  
<w type="nn0" teiform="w">people</w>  
<c type="pun" teiform="c">, </c>  
<w type="vrb-nn1" teiform="w">structure </w>  
<w type="dps" teiform="w">their </w>  
<w type="nn2" teiform="w">expectations </w>  
<w type="prf" teiform="w">of </w>  
<w type="pnx" teiform="w">themselves</w>  
<c type="pun" teiform="c">, </c>  
<w type="vrb" teiform="w">prevent </w>  
<w type="pnp" teiform="w">them </w>  
<w type="vvg" teiform="w">achieving </w>  
<w type="dps" teiform="w">their </w>  
...  
<c type="pun" teiform="c">.</c></s>
```

# XML for syntactic trees: TigerXML (WSJ)

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<corpus xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/home/compling/TIGERSearch/1.1rc2/schema/TigerXML.xsd" id="temp" >
  <head external="file:temp_generated_header.xml"/>
  <body>
    <s id="s1" >
      <graph root="s1_500" >
        <terminals>
          <t id="s1_1" word="Pierre" pos="NNP" />
          <t id="s1_2" word="Vinken" pos="NNP" />
          <t id="s1_3" word="," pos="," />
          <t id="s1_4" word="61" pos="CD" />
          <t id="s1_5" word="years" pos="NNS" />
          <t id="s1_6" word="old" pos="JJ" />
          <t id="s1_7" word="," pos="," />
          <t id="s1_8" word="will" pos="MD" />
          <t id="s1_9" word="join" pos="VB" />
          <t id="s1_10" word="the" pos="DT" />
          <t id="s1_11" word="board" pos="NN" />
          <t id="s1_12" word="as" pos="IN" />
          <t id="s1_13" word="a" pos="DT" />
          <t id="s1_14" word="nonexecutive" pos="JJ" />
          <t id="s1_15" word="director" pos="NN" />
          <t id="s1_16" word="Nov." pos="NNP" />
          <t id="s1_17" word="29" pos="CD" />
          <t id="s1_18" word="." pos="." />
        </terminals>
      </graph>
    </s>
  </body>
</corpus>
```

Corpus Linguistics

Linguistic  
Annotation

Corpus markup

Mark-up schemes  
Character encodings

Corpus annotation

Motivation  
How to annotate  
Levels of Annotation

Text Processing

Tokenization  
Lemmatization

Standalone  
annotation

More on XML

# WSJ in TigerXML (cont.)

```
<nonterminals>
  <nt id="s1_502" cat="NP" >
    <edge idref="s1_1" label="--" />
    <edge idref="s1_2" label="--" />
  </nt>
  <nt id="s1_504" cat="NP" >
    <edge idref="s1_4" label="--" />
    <edge idref="s1_5" label="--" />
  </nt>
  ...
  <nt id="s1_501" cat="NP" >
    <edge idref="s1_502" label="--" />
    <edge idref="s1_3" label="--" />
    <edge idref="s1_503" label="--" />
    <edge idref="s1_7" label="--" />
  </nt>
  ...
  <nt id="s1_500" cat="S" >
    <edge idref="s1_501" label="SBJ" />
    <edge idref="s1_505" label="--" />
    <edge idref="s1_18" label="--" />
  </nt>
</nonterminals>
</graph>
</s>
```

# Header information: define your classes

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<head>
  <meta>
    <name>/home/dickinso/temp.xml.gz</name>
    <format>bracketing format</format>
  </meta>
  <annotation>
    <feature name="word" domain="T" />
    <feature name="pos" domain="T">
      <value name="CC" />
      ...
    </feature>
    <feature name="cat" domain="NT">
      <value name="ADJP" />
      ...
    </feature>
    <edgelabel>
      <value name="--" />
      <value name="ADV" />
      ...
    </edgelabel>
    <secedgelabel>
      <value name="*" />
      ...
    </secedgelabel>
  </annotation>
</head>
```