User Guide for MaltEval 1.0 (beta)

Jens Nilsson
School of Mathematics and System Engineering
Växjö University 35195 Växjö, Sweden
`jens.nilsson@vxu.se`

October 5, 2014

Contents

1 Introduction

This is a user guide for MaltEval 1.0, an evaluation software for dependency parsing. MaltEval has been developed at Växjö University, and is freely available for any purposes. It comes with no guarantees and is distributed “as is”. It has been created to make evaluation and visualization of dependency structure easier. Moreover, some functionality for extending MaltEval using plugins has been implemented in order to increase its the usability, as no evaluator can contain every evaluation type that every user could need.

2 Run MaltEval

MaltEval requires Java JRE 1.6[1] or higher in order to function. It is executed by typing:

```
java -jar MaltEval.jar
```

A welcome text and information concerning the usage of MaltEval and some of its arguments should then appear on the screen. The displayed information shows three types of arguments that are available in the current version of MaltEval, the required and optional arguments (this section), as well as other arguments (section 3). There is also a forth important type of argument, the evaluation arguments (section 4), which are shown by typing:

```
java -jar MaltEval.jar --help
```

This output lists all arguments that can manipulate the evaluation in numerous ways. Using the `examples`-flag like this

```
java -jar MaltEval.jar --examples
```

examples of how to control the evaluation in different ways are shown.

The usage of MaltEval looks like this:

```
java -jar MaltEval.jar
    [-e <evaluation file> (optional)] [arguments]
```

The first argument `-e <evaluation file>` is optional. The evaluation file argument can only be located as the first argument of MaltEval. The value after `-e` is the path to an evaluation file in an XML format discussed later on. Most things that can be specified as an argument in `[arguments]` can also be specified in the evaluation file. In case an argument is specified twice, the argument’s value in the evaluation file will be overridden.

[1]http://www.java.com/en/download/index.jsp

There are a number of arguments that can only be specified through flags to MaltEval directly, and not in an evaluation file. They are divided into required and optional flags, discussed in the two coming subsections (2.1 and 2.2).

2.1 Required flags

There are two required flags that specify the gold-standard file(s) and the file(s) that you want to evaluate:

```
-g <gold-standard file, files or directory (tab|xml|conll)>
-s <file, files or directory with parser output (tab|xml|conll)>
```

The file format can be either the CoNLL-format (appendix B.1), MaltXML-format (appendix B.2) or MaltTab-format (appendix B.3), where the CoNLL-format is the default format. Any file with the extension `.xml` or `.tab` is interpreted as an MaltXML and MaltTab file, respectively, while any other extension is considered to be formatted according to the CoNLL-format, usually having the extension `.conll`.

2.1.1 Single File Evaluation

The flags `-s` and `-g` can be followed by a single file each, e.g.:

```
java -jar MaltEval.jar -s parser.conll -g gold.conll
```

In this case, where no evaluation file and evaluation flags are specified, simple comparison of the two files is done using the default evaluation settings. The output (on the standard output) should look similar to this:

```
===================================
Gold:   gold.conll
Parsed: parser.conll
===================================
GroupBy-> Token
Metric-> LAS

===================================

accuracy   token
------------------------
0.867      Row mean
70162      Row count
------------------------
```

The names of the gold-standard and parsed files are shown, followed by the information that MaltEval grouped the output by individual tokens and that the metric LAS (Labeled Attachment Score) was used. The small table then presents that the accuracy is 86.7% and that the test data contains 70162 distinct group instances. In this case, each group instance corresponds to an individual token, entailing that there are 70162 tokens.

2.1.2 Multiple Files Evaluation

If you want to compare more than one parsed file using the same evaluation settings and gold-standard file, the parsed files can simply be listed after `-s` like this:

```
java -jar MaltEval.jar
   -s parser1.conll parser2.conll parser3.conll -g gold.conll
```

This could be helpful for several reasons, such as comparing different parsers or when you want to evaluate a learning curve experiment where the same test data has been used for one parser with different amounts of training data.

2.1.3 Cross Validation Evaluation

It is also possible to automatically compute mean for cross validation experiments by evoking MaltEval in the following way:

```
java -jar MaltEval.jar
   -s p_set1.conll p_set2.conll p_set3.conll
   -g g_set1.conll g_set2.conll g_set3.conll
```

Here `p_set1.conll` is compared to `g_set1.conll` and `p_set2.conll` is compared to `g_set2.conll` and so on, where the number of files after `-s` and `-g` must be the same in order to perform an evaluation. This is in contrast to Multiple Files Evaluation above, where multiple parsed files were compared to a single gold-standard file. It is important that the number of parsed files equals the number of gold-standard files when there are more than one gold-standard file, since the evaluation otherwise is aborted.

2.1.4 Wild Card and File Sorting

The lists of files for `-s` and `-g` can grow very long, which makes them tedious to type. If you are using a shell, you can make use of its utility to expand paths using wild cards. So instead of typing e.g.
`-s parser1.conll parser2.conll parser3.conll`, you can for instance type `-s parser?.conll` or `-s parser*` depending on the content of the directory.

It is important to remember that the files are ordered alphabetically before the evaluation takes place. For example, with the intention to compare `p_set2.conll` with `g_set1.conll` and `p_set1.conll` with `g_set2.conll`, one **cannot** type:

```
java -jar MaltEval.jar
   -s p_set2.conll p_set1.conll
   -g g_set1.conll g_set2.conll
```

The parsed files are sorted before evaluation, hence comparing `p_set1.conll` to `g_set1.conll` and `p_set2.conll` to `g_set2.conll`. The reason for sorting is that one may end up in problems in some shells when using wild card symbols, as the order of the expanded list of files may differ. The only work-around is to rename the files.

2.1.5 Using Directories as Arguments

It is also possible to instead type the name of a directory for either `-s` or `-g`, or both, e.g.:

```
java -jar MaltEval.jar
   -s parsedDir/
   -g goldDir/
```

In this particular situation, all files with any of the extensions `.xml`, `.tab` or `.conll` located in the directories are sorted alphabetically. Depending on the number of such files in each directory, either single, multiple or cross validation evaluation is performed.

2.2 Optional flags

MaltEval has three optional flags, one for specifying a character set encoding, and two for files containing the lists of parts-of-speech and dependency types.

2.2.1 The `charset` flag

The flag `--charset` makes it possible to alter the character set encoding. This flag is only applicable to the CoNLL and MaltTab formats, since the character set is specified directly in the MaltXML format. The default character set is UTF-8, but if the gold-standard and parsed files are encoded in for instance ISO-8859-1, then type the following instead (where the order of the flags is irrelevant):

```
java -jar MaltEval.jar --charset ISO-8859-1
   -s parser.conll -g gold.conll
```

2.2.2 The Validation flags

By default, no validation in done before the evaluation for the dependency types and part-of-speech tags for files in the CoNLL or MaltTab format. It is possible to perform a test to see whether the POSTAG or DEPREL attribute for each token has a valid value by specifying a file containing the complete set of valid part-of-speech tags or deprel types. This is done in the following way:

```
java -jar MaltEval.jar -s parser.conll -g gold.conll
   --postag gold.postag --deprel gold.deprel
```

The files `gold.postag` and `gold.deprel` are text files and contain the sets of valid part-of-speech tags and dependency type, with one tag/type per line.[2] MaltEval terminates with an error message if any input file contains invalid tags/types.

2.2.3 The Tree Viewer Flag

MaltEval has a module for viewing the content of the gold-standard and parsed files visually. The visual mode is enabled by setting the flag `-v` to 1 (default is 0). This also disables all other flags except `-s` and `-g`, and will therefore not perform any evaluation in the normal sense.

One can for instance type

```
java -jar MaltEval.jar -v 1
   -s parsed1.conll parsed2.conll -g gold.conll
```

in order to create a window depicting the content of the three files one sentence at a time, with a list of all sentences for changing which dependency trees to show. Figure 1 in appendix C shows an example of how such a window can look like. [3]

The requirement that both `-s` and `-g` must be specified is relaxed if the visualization mode is enabled. In this case, only one of them needs to be specified. One can type `java -jar MaltEval.jar -v 1 -s parsed1.conll` or `java -jar MaltEval.jar -v 1 -g gold.conll` to visualize just one file.

The visualization module also comes with a search tool, which is based on the grouping strategy. This can also be seen in figure 1.

3 Evaluation Settings

It is possible to perform other types of evaluation than just the default labeled attachment score. The evaluation settings can be modified in two ways, either by using an evaluation file or by adding evaluation arguments to MaltEval directly. They are discussed in the two subsections below (3.1 and 3.2).

[2]Empty lines anywhere in the file must be removed

[3]For simplicity, it is only possible to specify at most one gold-standard file, see Cross Validation Evaluation (2.1.3), since the list of sentence can only be connected to one gold-standard file.

3.1 Evaluation File Argument

A simple evaluation file (default.xml) could look like this:

```
<evaluation>
  <parameter name="Metric">
    <value>LAS</value>
  </parameter>
  <parameter name="GroupBy">
    <value>Token</value>
  </parameter>
</evaluation>
```

The root element is named *evaluation* and contains a list of zero or more *parameter* elements. Each parameter element has a required *name* attribute, containing the name of the parameter to set.

Each parameter has a default value which is overridden if it is specified in the evaluation file. New values for a parameter are added using zero or more *value* elements located under the parameter element. In the example we can see that `LAS` is added to *Metric* and that `Token` is added to *GroupBy*, which corresponds to the default settings. The evaluation in subsection 2.1.1 is hence an abbreviation of:

```
java -jar MaltEval.jar -e default.xml
   -s parser.conll -g gold.conll
```

The semantics of the evaluation file is presented in 3.1.1 below in this subsection.

The available parameters are: `Metric`, `GroupBy`, `MinSentenceLength`, `MaxSentenceLength`, `ExcludeWordforms`, `ExcludeLemmas`, `ExcludeCpostags`, `ExcludePostags`, `ExcludeFeats`, `ExcludeDeprels`, `ExcludePdeprels` and `ExcludeUnicodePunc`.[4] The parameters `Metric` and `GroupBy` have restricted sets of possible values, enumerated in 3.1.2 and 3.1.3, whereas no control of the values is performed by MaltEval for the others.

3.1.1 The Semantics of the Evaluation File

As the evaluation element consists of a list of parameters and each parameter consists of a list of value elements, the generic evaluation file format looks like this:

```
<evaluation>
  <parameter name="par1">
    <value>par1_val1</value>
    <value>par1_val2</value>
    <value>...</value>
  </parameter>
```

[4]The name attribute is sensitive to case in the evaluation file.

```
  <parameter name="par2">
    <value>par2_val1</value>
    <value>par2_val2</value>
    <value>...</value>
  </parameter>
  ...
</evaluation>
```

The parameter list is treated as a set with the name attribute as the key, where the last parameter element with a distinct key overrides all previous parameter elements with the same key. Each list of value elements is also treated as a set. MaltEval then performs one evaluation for every possible combination of values for all parameters. For instance, with two parameters having three and four values, respectively, twelve evaluations will be computed by MaltEval by combining every value of the first parameter with every value of the second parameter.

MaltEval tries to merge all results for each parsed file into one table whenever it is suitable. With, for instance,

```
<parameter name="Metric">
  <value>LAS</value><value>UAS</value><value>LA</value>
</parameter>
```

a table with three columns is created, one for each `Metric`-value. However, some combinations would result in strange merged tables. Specifically, for multiple `GroupBy`-values, tables will not be merged due to different types and number of rows. MaltEval therefore presents the output is separate tables instead. See also `merge-tables` in subsection 4.1.10 about how to manipulate the merging strategy.

3.1.2 The `Metric` Values

The currently available values for `Metric` are shown below, where two different values can be used for the first three:

`LAS` (BothRight) A token is counted as a hit if both the head and the dependency label are the same as in the gold-standard data. This is the default value.

`LA` (LabelRight) A token is counted as a hit if the dependency label is the same as in the gold-standard data.

`UAS` (HeadRight) A token is counted as a hit if the head is the same as in the gold-standard data.

`AnyRight` A token is counted as a hit if either the head or the dependency label (or both) is the same as in the gold-standard data.

`BothWrong` A token is counted as a hit if neither the head nor the dependency label are the same as in the gold-standard data.

`LabelWrong` A token is counted as a hit if the dependency label is **not** the same as in the gold-standard data.

`HeadWrong` A token is counted as a hit if the head is **not** the same as in the gold-standard data.

`AnyWrong` A token is counted as a hit if either the head or the dependency label (or both) is **not** the same as in the gold-standard data.

`self` This is a special type of metric that is dependent on the selected `GroupBy` values (see 3.1.3 below). Each grouping strategy has a of called self metric which is applied when the metric value equals `self`. The self value is applicable for all grouping strategies but is in practice only useful for grouping strategies where the grouping values of the gold-standard data and the parsed data may differ. For instance, if one specify `self` as the metric for the grouping strategy ArcProjectivity (see paragraph `ArcProjectivity` in 3.1.3), one will use ArcProjectivity both as the metric and the grouping strategy. The output for

```
java -jar MaltEval.jar --Metric self
   --GroupBy ArcProjectivity -g gold.conll -s parser.conll
```

could look like this:

```
====================================================
Metric-> ArcProjectivity
GroupBy-> ArcProjectivity

====================================================

precision   recall     fscore   ArcProjectivity
-------------------------------------------------
0.03        0.07       0.04     Non-proj
0.99        0.97       0.98     Proj
```

For instance, here we can see that 7% of all non-projective tokens in the *gold-standard* data are non-projective in the parsed data according to the gold-standard.

Also, 3% of all non-projective tokens in the *parsed data* are non-projective according to the gold-standard. The self value enables the evaluator to compute fscore as well.

The `Token` and `Postag` grouping strategies are two examples where the grouping values of the gold-standard data and the parsed data may not differ.

3.1.3 The `GroupBy` Values

Any `Metric` value can be combined with any `GroupBy` value. The average (Row mean) in the output depends on the grouping strategy, since the tokens are grouped before the row mean is computed. The standard approach when comparing the average accuracy in dependency parsing is obtained for the default `Token` value.[5] All grouping strategies are listed below, including a short description. Each grouping strategy below ends with a list of all individual `format` attributes. This is described in subsection 3.1.5 and is used to further control the type of information that will be presented in the output.

`Token` Each token is treated individually. The mean value is therefore computed by dividing the number of tokens with a hit (according to the `Metric` value) with the total number of tokens in the data, the standard way of computing accuracy in dependency parsing.

Available values for the format attribute: accuracy.

`Wordform` All tokens with the same value for the wordform attribute (case sensitive) are grouped together.

`Lemma` All tokens with the same value for the lemma attribute (case sensitive) are grouped together.

`Cpostag` All tokens with the same value for the cpostag attribute (case sensitive) are grouped together.

`Postag` All tokens with the same value for the postag attribute (case sensitive) are grouped together.

[5]Note therefore that the row mean seldom makes sense for any other grouping strategy than `Token`. One possible exception is the `Sentence` grouping, where the metric is computed as the mean of all sentence means, tending to render slightly higher mean than for `Token`.

`Feats` All tokens with the same value for the feats attribute (case sensitive) are grouped together. Each feat value is therefore treated as an atomic value.

`Deprel` All tokens with the same value for the deprel attribute (case sensitive) are grouped together.

`Sentence` All tokens in the same sentence are treated as one group.

`RelationLength` All tokens with the same arc length are grouped. Arc length is computed as the (positive) difference in word position between a token and the token's head. Hence, whether the dependent is located to the left or right of the head is indifferent. Root tokens, i.e. tokens without heads, are treated as one group separately having the value -1. The value 0 is reserved for any tokens having an arc pointing to itself.

`GroupedRelationLength` This is a similar grouping strategy as `RelationLength`, where the difference is that the lengths are grouped into either "to_root", "1", "2", "3–6" or "7–...".

`SentenceLength` The tokens are grouped according to sentence length, which can be any integer value equal or greater than 1.

`StartWordPosition` This strategy groups tokens according to the tokens' positions in the sentence counted from the sentence start. That is, the first token of every sentence belongs to group 1, the second token to group 2, and so forth.

`EndWordPosition` The opposite to `StartWordPosition`, e.g. the last token of each sentence belongs to group 1, the second last token belongs to group 2, and so forth.

`ArcDirection` Each token is mapped to one of four values, depending on the direction of the arc. The group "left" contains all token with the head located to the left of itself, and the group "right" then contains all token with the head located to the right of itself. All root tokens are treated separately as the group "to_root". All tokens with itself as the head is mapped to the group "self", which hopefully is an empty group.

`ArcDepth` The tokens are groups according the distance to the root token. Again, all tokens with out a head token are grouped separately, in this case in group “0”, and all tokens on depth 1 from the root will consequently form the group “1”, and so forth.

`BranchingFactor` This grouping strategy is the number of direct dependents of a token as key for grouping, which can be any integer value equal or greater than 0.

`ArcProjectivity` This grouping strategy has only two values, “0” and “1” representing projective and non-projective arcs. [6]. Informally, an arc is projective if all tokens it covers are descendants of the arc’s head token.

`Frame` For this grouping strategy, the dependency labels of a token and its dependents are used. The dependency types of the dependents are sorted according their position in the sentence, separated by a white space. The dependency label of the token’s dependency label is positioned between the left and right dependents surrounded by two *-characters.

For example, a token with the dependency label *Pred* having a *Sub* dependent to the left, and an *Obj* dependent followed by an *Adv* dependent to the right, would be one instance of the frame group *Sub *Pred* Obj Adv*. Note that the evaluation is computed for the token with the dependency label *Pred* and/or its head value, not the complete frame.

3.1.4 Complex `GroupBy` Values

The complex grouping is a generalization of the simple `GroupBy` values. The complex `GroupBy` values are currently supported for the simple `GroupBy` values `Wordform`, `Lemma`, `Cpostag`, `Postag` and `Feats`. It is possible to create groups by combining the simple ones and to group according to context of the focus word. Here is the syntax of the complex grouping:

```
ComplexGroup ::= SingleGroup ( ; SingleGroup )*
SingleGroup  ::= SimpleValue ( @ RelPos )?
SimpleValue  ::= Wordform | Lemma | Cpostag | Postag | Feats
RelPos       ::= ..., -2, -1, 0, 1, 2, ...
```

[6]The definition of non-projectivity can be found in Kahane, S., A. Nasr, and O. Rambow (1998). Pseudo-Projectivity: A Polynomially Parsable Non-Projective Dependency Grammar. In Proceedings of COLING/ACL.

A complex group consists of one or more SingleGroups separated by semicolons. A SingleGroup decomposes into one of the five SimpleValues and an optional RelPos separated by @. Absent @ and RelPos is the same as typing “@0”. RelPos can be any positive or negative integer value.

The semantics is that each ComplexGroup is a set of SingleGroup items consisting of the pair SimpleValue/RelPos. Position 0 for RelPos represents the token for which the metric is computed, i.e. the focus word. A negative or positive position does instead look at a token to the left or right, respectively, of the focus word. In other words,

```
<value>Wordform@0</value>
```

means that the result is grouped according the focus word, equivalent to the simple grouping

```
<value>Wordform</value>
```

and

```
<value>Wordform@-2</value>
```

group according to the wordform of the token two steps to the left of the focus word.

The items of a ComplexGroup are combined by conjunction. For instance,

```
<value>Cpostag@0;Wordform@0</value>
```

means that the cpostag value and wordform value of the focus word together form a group instance. The value

```
<value>Cpostag@-2;Cpostag@-1</value>
```

instead forms a group of the cpostag values of the two preceding tokens.

In cases where the evaluator looks outside the sentence, the characters ˆ (start of sentence) and $ (end of sentence) will be part of the grouping instances in the evaluation output. For instance, having the second token of sentence as focus word when looking at the two preceding cpostags, where the cpostag of the first word is *Det*, the group instance is `ˆ@-2;Det@-1`.

3.1.5 The `format` Attribute: Select, Sort and Cut

The format attribute of the value element is used to control the type of information that is selected to be displayed as columns in the output. In case the optional format attribute is not specified for a value for a `GroupBy` parameter, or equals the empty string, the default format attributes are selected, which is either accuracy, or precision and recall. However, all available attributes will be displayed by typing i.e.

```
<parameter name="GroupBy">
   <value format="all">Sentence</value>
</parameter>
```

in the evaluation file.

Select Some grouping strategies, such as `Sentence`, displays a large amount of format information for `all`, so the format attribute can be used for selecting a subset of the information, separated by |, in the following way:

```
<value format="accuracy|correctcount|sentencelength">
   Sentence
</value>
```

Here is the list of available format attributes and a short description of their meanings:

- Applicable to all of them (including any complex grouping) except `Token`:
  - `correctcounter`: the number of tokens that were correct in a group according to the `Metric` value.
- Applicable to `Wordform`, `Lemma`, `Cpostag`, `Postag`, `Feats`, `Sentence`, `SentenceLength`,`StartWordPosition`, `EndWordPosition` and any complex grouping:
  - `counter`: the number of tokens in a group.
- Applicable to `Token`, `Wordform`, `Lemma`, `Cpostag`, `Postag`, `Feats`, `Sentence`, `SentenceLength`, `StartWordPosition`, `EndWordPosition` and any complex grouping:
  - `accuracy`: correctcounter divided by counter according to the `Metric` value. This is simply 0 or 1 for `Token`.
- Applicable to `Sentence`:
  - `exactmatch`: were all included tokens in the sentence correct according to the `Metric` value? 0 or 1.
  - `includedtokenscount`: the number of tokens included in the sentence.
  - `sentencelength`: the number of tokens in the sentence; equals includedtokenscount if no tokens are excluded (see subsection 3.1.7).

  - `isparserconnected`: does the dependency graph in the parser data have more than one root? 0 or 1.
  - `istreebankconnected`: does the dependency graph in the gold-standard data have more than one root? 0 or 1.
  - `hasparsercycle`: does the dependency graph in the parser data contain at least one cycle? 0 or 1.
  - `hasparsercycle`: does the dependency graph in the gold-standard data contain at least one cycle? 0 or 1.
  - `isparserprojective`: is the dependency graph in the parser data projective? 0 or 1.
  - `istreebankprojective`: is the dependency graph in the gold-standard data projective? 0 or 1.

- Applicable to e.g. `Deprel`, `ArcLength`, `ArcDirection`, `ArcDepth`, `Frame`, `BranchingFactor` and `ArcProjectivity` for the `self` metric (see 3.1.2):

  - `parsercount`: the number of tokens in a group according to the parser data.
  - `treebankcount`: the number of tokens in a group according to the gold-standard data.
  - `precision`: correctcounter divided by parsercount.
  - `recall` correctcounter divided by treebankcount.
  - `fscore` the harmonic mean of precision and recall, i.e.:
    F-score = (2 × `precision` × `recall`)/(`precision` + `recall`).

- Applicable to e.g. `Deprel`, `ArcLength`, `ArcDirection`, `ArcDepth`, `Frame`, `BranchingFactor` and `ArcProjectivity` for other than the self metric (see 3.1.2):

  - `parsercounter`: the number of tokens with a given grouping value according in the parsed data.

  - `treebankcounter`: the number of tokens with a given grouping value according in the gold-standard data.

  - `parsercorrectcounter`: the number of correct tokens with a given grouping value according in the parsed data.

- `treebankcorrectcounter`: the number of correct tokens with a given grouping value according in the gold-standard data.

- `parseraccuracy`: parsercorrectcounter divided by parsercounter

- `treebankaccuracy`: treebankcorrectcounter divided by treebankcounter

Sort The format attribute is also used for sorting the output according to a certain column of the output, but it only makes sense when the formatting argument `details` is enabled (see section 4.1.2). If one, for example, wants to sort the output in ascending order according to accuracy using the `Postag` grouping, then type

```
<value format="accuracy+">Postag</value>
```

and in descending order like this

```
<value format="accuracy-|correctcount">Postag</value>
```

where the correctcount for each deprel is also displayed. If one wants to display all available information and sort according to accuracy, one can type:

```
<value format="all|accuracy-">Postag</value>
```

It is only possible to sort by a single attribute, so in case more than one attribute is suffixed with - or +, the last sorting overrides all previous ones.

Cut For some grouping strategies, the number of distinct groups may be very large, such as grouping by sentence. If the formatting argument `details` is enabled, the amount of output easily becomes unwieldy. By typing a positive integer value after the - or +, i.e.:

```
<value format="all|accuracy-20">Postag</value>
```

only the 20 dependency type with the highest accuracy are shown in the output. Note that the mean value is still computed for all deprels.

3.1.6 The `...SentenceLength` Values

To restrict the evaluation to a certain interval for sentence length, the `MinSentenceLength` and `MaxSentenceLength` are used. The value is a natural number ($0, 1, 2, \ldots$). Both have the default value 0, which represents positive infinity for `MaxSentenceLength`. The values are included in the interval.

For instance,

```
<parameter name="MinSentenceLength">
   <value>1</value>
</parameter>
<parameter name="MaxSentenceLength">
   <value>40</value>
</parameter>
```

means that only sentences of length $1 \ldots 40$ (including both 1 and 40) are evaluated.

3.1.7 The `Exclude...` Values

In some cases, one wants to exclude some tokens from the evaluation, such as punctuation. This is generalized in such a way that any set of values for wordform, lemma, cpostag, postag, feats, deprels and pdeprels can be excluded. If one wants to exclude all tokens having the dependency label *Punc*, then add a parameter element to the evaluation file looking like this (where the name attribute is case sensitive):

```
<parameter name="ExcludeDeprels">
   <value>Punc</value>
</parameter>
```

More than one value can be excluded by separating dependency labels by $|$, i.e.:

```
<parameter name="ExcludeDeprels">
   <value>Punc|Sub</value>
</parameter>
```

Two or more `Exclude...` parameters can be combined by disjunction:

```
<parameter name="ExcludeWordforms">
   <value>.</value>
</parameter>
<parameter name="ExcludeDeprels">
   <value>Punc</value>
</parameter>
```

This means that a token is excluded if it has either the wordform . **or** the dependency type *Punc* (or both).

Hint: if you want to evaluate your result both with and without the dependency label *Punc* (without creating two evaluation files), then type (see also subsection 3.1):

```
<parameter name="ExcludeDeprels">
   <value></value>
   <value>Punc</value>
</parameter>
```

3.1.8 The `ExcludeUnicodePunc` Values

For compatibility with the evaluation script eval.pl[7], the special exclude parameter `ExcludeUnicodePunc` is provided. Just as eval.pl, MaltEval is able to exclude tokens where all characters of its wordform have the Unicode property "punctuation". Read the documentation for eval.pl for information about which characters have this property.

This parameter is also combined by disjunction together with other `Exclude`-parameters. It can have the value "0" or "1", wherethe latter obviously enables `ExcludeUnicodePunc`. Default is "0".

Note again that this parameter is mainly added for compatibility reasons. It has unfortunately some side effects, such as excluding some tokens that you normally do not want to exclude, and including some tokens that you normally do not want to include.

3.2 Evaluation Flags

All evaluation parameters, discussed in subsection 3.1, that can be altered using an evaluation file, can also be altered using MaltEval flags directly. The corresponding flag for a parameter name, i.e. `Metric`, `GroupBy`, `...SentenceLength`, or `Exclude...`, is constructed by adding the prefix `--`. The value(s) of a parameter is located after the flag separated by a space character, e.g.:

```
 --Metric LAS
```

Multiple values are separated by a semicolon (no additional spaces are allowed), meaning that

```
<parameter name="ExcludeDeprels">
   <value>Dep1</value>
   <value>Dep2</value>
</parameter>
```

corresponds to `--ExcludeDeprels Dep1;Dep2`[8], where either `Dep1` or `Dep2` can be an empty string (e.g. `--ExcludeDeprels ;Punc` corresponds to the last example in subsection 3.1.7).

The format attribute (used only by `GroupBy`) to manipulate selecting, sorting and cutting can be suffixed to a value using a colon. Hence, the parameter element

```
<parameter name="GroupBy">
   <value format="all|parseraccuracy-20">Deprel</value>
</parameter>
```

[7]http://nextens.uvt.nl/~conll/software.html#eval

[8]Keep in mind that many shells use semicolon to serialize shell commands, making it necessary to surround the values in quotes: `--ExcludeDeprels "Dep1;Dep2"`.

yields the same output as using the flag

```
 --GroupBy Deprel:all|parseraccuracy-20
```

This is an example of a valid usage of flags for MaltEval:

```
java -jar MaltEval.jar --Metric LAS;UAS;LA
   --GroupBy Postag:all|accuracy-20;Deprel:parseraccuracy+
   --ExcludeLemmas this|that
   -s parser.conll -g gold.conll
```

This call results in six evaluations in sequence, since `--Metric` has three values (separated by semicolons) and `--GroupBy` two, all of them excluding any token having the lemma value *this* or *that*.

4 Formatting Settings

The formatting arguments can be specified in the same two ways as the evaluation settings. The first subsection (4.1) below will describe how this is done using the evaluation file, while the following subsection (4.2) presents how the same behavior can be achieved using flags instead.

4.1 Formatting Arguments

There are currently nine formatting arguments that change how the evaluation is formatted or presented. The formatting elements in the evaluation file looks like this:

```
<formatting argument="..." format="..."/>
```

Any formatting element can be located before, between or after any parameter element in the evaluation file. The required argument attribute is occupied by any of the format names listed in the subsections below (4.1.2–4.1.9). The value of the required format attribute is dependent on the value of the argument attribute, also described below.

4.1.1 The `micro-average` Formatting

The format value can be either 0 or 1. This formatting is only applicable during cross validation, i.e. when two or more gold-standard file and equally many parsed files have been specified. In case this value equals 1, the gold-standard files are merged into one file before evaluation, as well as the parsed files. The gold-standard files and parsed files, respectively, are then treated just as they had been two files. We call this the micro-average.

If the value is 0, each pair of gold-standard file and corresponding parsed file is evaluated separately, and the average is computed after the evaluation of all pairs. We call this the macro-average.

The default value is 0.

4.1.2 The `details` Formatting

The `format` value can be either 0 or 1. It specifies whether all distinct groups are displayed in the output or just the row mean and row count. If disabled, the output for `--GroupBy ArcDepth:parseraccuracy` could look for instance like this:

```
precision   Arcdepth
---------------------
0.852       Row mean
14          Row count
---------------------
```

If enabled, is instead looks like this:

```
parseraccuracy   Arcdepth
---------------------
0.852            Row mean
14               Row count
--------------------------
0.864            0
0.855            1
0.78             2
0.793            3
0.793            4
0.803            5
0.794            6
0.822            7
0.801            8
0.872            9
0.75             10
1                11
1                12
1                13
```

The default value is 0, but each grouping strategy specifies whether the details should be displayed or not. If the detail flag is explicitly set, then the default details value of any grouping strategy is overridden.

4.1.3 The `header-info` Formatting

The `format` value can be either 0 or 1. The formatting makes is possible to enable or disable the header info of each table in the output, including column headers, row mean and row count. Unless the `details`, `stat` or `confusion-matrix` formatting is enabled (see below), the evaluator produces no output at all.

The default value is 1.

4.1.4 The `row-header` Formatting

The `format` value can be either 0 or 1. Displays the row headers or not. With `row-header` disabled and `details` enabled, the output could be:

```
parseraccuracy
--------------
0.852
14
--------------
0.864
0.855
0.78
0.793
0.793
0.803
0.794
0.822
0.801
0.872
0.75
1
1
1
```

Hint: by disabling both `header-info` and `row-header` and enabling `details`, the output will just contain numbers, one per line. This format is sometimes suitable input format for various tools computing statistical significance, if the statistical significance tests incorporated in MaltEval is not applicable.

The default value is 1.

4.1.5 The `tab` Formatting

The `format` value can be either 0 or 1. Specifies whether the columns in the output are separated by tab (1) or multiple spaces (0). If `tab` is disabled, the number of added spaces between the columns depends on the lengths of the cell values to make the output as pretty as possible in a text file. Tabs may be more suitable when importing the output to for instance chart programs or Excel.

The default value is 0.

4.1.6 The `output` Formatting

The `format` value is either STDOUT (default), a path to a file or a directory. If STDOUT is chosen, all output is simply sent to the standard output stream (the screen). If a file is chosen, everything it is instead printed to that file.

In case a directory is specified, the output is distributed to different files depending on whether multiple evaluation or cross validation is performed. If `stat` is enabled (see below), the statistics is written to a separate file as well.

4.1.7 The `pattern` Formatting

By default, every floating point value in the output is printed with three decimals (using the pattern `0.000`). The `pattern` formatting changes this. See http://java.sun.com/j2se/1.5.0/docs/api/java/text/DecimalFormat.html for a description of the syntax and semantics.

4.1.8 The `stat` Formatting

The `format` value can be either 0 or 1. Currently only McNemar's test is implemented, which makes it applicable only to columns in the output having no other values that 0s and 1s. In order to make sense at least two parsed files must be specified. If more than two parsed files are specified, the McNemar's test is applied pairwise between all parsed data sets.

McNemar's test is for instance applicable to `--GroupBy Token:accuracy`, since each accuracy values for the `Token` grouping is either 0 or 1. For three parsed files (parsed1.conll, parsed2.conll, parsed3.conll), the statistical significance result could look like this:

```
GroupBy-> Token:accuracy

Attribute: accuracy
<1>   <2>     <3>     McNemar: z-value
-----------------------------------------
-     2.768   0.323   <1> (parsed1.conll)
-     -       2.912   <2> (parsed2.conll)
-     -       -       <3> (parsed3.conll)

<1>   <2>   <3>   McNemar: p<0.01?
--------------------------------------
-     1     0     <1> (parsed1.conll)
-     -     1     <2> (parsed2.conll)
-     -     -     <3> (parsed3.conll)

<1>   <2>   <3>   McNemar: p<0.05?
--------------------------------------
-     1     0     <1> (parsed1.conll)
-     -     1     <2> (parsed2.conll)
-     -     -     <3> (parsed3.conll)
```

The first table shows the z-value for the accuracy for the token grouping, and the two others what the z-value corresponds to in terms of statistical significance levels. For instance, the 1 in column <2> and row <1> in the last table means that there is a statistically significant difference between parsed1.conll and parsed2.conll beyond the .05 level.

For `--GroupBy Sentence:accuracy`, McNemar's test is not applicable since the accuracies for sentence could be any floating value between 0 and 1. However, `--GroupBy Sentence:exactMatch|isProjective` produces statistical significance tables for both exactMatch and isParserProjective, but whether the result is useful is up to the user to assess (which is questionable for isParserProjective).

The default value is 0.

4.1.9 The `confusion-matrix` Formatting

The `format` value can be either 0 or 1. MaltEval is able to produce confusion matrices and confusion tables if `confusion-matrix` is enabled. It is applicable to any `GroupBy` value, but it only makes sense for those which can compute precision and recall (i.e. `GroupBy` values `Deprel`, `ArcLength`, `ArcDirection`, `ArcDepth`, `Frame`, `BranchingFactor` and `ArcProjectivity`).

The confusion result is presented in two ways. MaltEval creates an ordinary confusion matrix if the number of group values for the parsed data multiplied by the number of group values for the treebank data is less than 2,500 (a 50x50 matrix). Large confusion matrices are simply not graspable for a human, and really large ones too computationally demanding. With the setting `--GroupBy ArcDepth`, the confusion matrix could look like this:

```
Confusion matrix for ArcDepth
0    1     2       3       4       5       6     7     8     9    10   11   12   13   Col: system
                                                                                     / Row: gold
-----------------------------------------------------------------------------------------------------
-    294   120     80      62      26      9     3     3     0    0    0    0    0    0
89   -     1459    457     188     101     53    11    6     4    0    0    0    0    1
21   926   -       2140    616     238     90    52    14    2    0    0    0    0    2
4    235   1360    -       1745    551     186   55    40    14   2    0    0    0    3
2    64    368     1176    -       1044    339   116   27    18   9    1    0    0    4
1    20    97      322     782     -       557   192   47    16   9    0    3    0    5
2    7     25      75      226     373     -     244   86    26   8    0    0    2    6
0    6     1       14      57      119     127   -     104   24   9    6    0    0    7
0    0     2       4       16      33      54    31    -     44   11   0    1    0    8
0    0     0       3       6       4       16    18    11    -    17   2    0    0    9
0    0     0       0       1       5       2     10    10    0    -    3    0    0    10
0    0     0       0       0       1       4     0     4     4    0    -    2    0    11
0    0     0       0       0       0       1     0     0     0    1    0    -    0    12
0    0     0       0       0       0       0     1     0     0    0    2    0    -    13
```

and for `--GroupBy ArcDirection` like this:

```
Confusion matrix for ArcDirection
left    right   to_root   Col: system / Row: gold
-------------------------------------------------
-       1581    64        left
1033    -       60        right
359     245     -         to_root
```

The column contains the parsed data values, while the correct values of the gold-standard data are shown in the rows. Besides the confusion matrix, a confusion table is always produced when `confusion-matrix` is enabled. It sorts the system/gold-pairs by frequency, which for `--GroupBy ArcDirection` could result in the output:

```
Confusion table for ArcDirection
frequency   System / Gold
---------------------------
1581        left / right
1033        right / left
359         to_root / left
245         to_root / right
64          left / to_root
60          right / to_root
```

As this table can grow very long, a threshold of 50 rows is applied. This is illustrated by the confusion table for `Frame` below, having as many as 50+7,076 frames pairs with a frequency above 0:

```
Confusion table for frame
frequency   System / Gold
----------------------------------------
95          *MNR* NK  / *MO* NK
72          *MO* NK  / *MNR* NK
66          *ROOT*  / *PUNC*
62          *MNR* NK NK  / *MO* NK NK
57          *SB*  / *OA*
56          *MO* NK NK  / *MNR* NK NK
53          MO *NK*  / *NK*
51          *OA*  / *DA*
50          *OA*  / *SB*
...
18          NK *SB* AG  / NK *SB*
18          *MNR* NK NK  / *OP* NK NK
17          *SB*  / *SB* RE
7076 more...
```

The default value is 0.

4.1.10 The `merge-tables` Formatting

The `format` value can be either 0 or 1. MaltEval tries, whenever possible, to merge all results into one table in the output when multiple values for one or more evaluation parameters have been specified. For instance, `--Metric LAS;UAS;LA` could result in:

```
====================================================
GroupBy-> Token

====================================================

accuracy / Metric:UAS   accuracy / Metric:LAS   Token
---------------------------------------------------------
0.8941                  0.8687                  Row mean
70162                   70162                   Row count
---------------------------------------------------------
```

This is the default behavior, but it is disabled if `merge-tables` is set to 0. The above result would then instead be presented as:

```
...
====================================================
Metric-> UAS
GroupBy-> Token

====================================================

accuracy   Token
--------------------
0.8941     Row mean
70162      Row count
--------------------
...
====================================================
Metric-> LAS
GroupBy-> Token

====================================================

accuracy   Token
--------------------
0.8687     Row mean
70162      Row count
--------------------
```

The default value is 1.

4.2 Formatting Flags

The formatting arguments in the preceding subsection (4.1) for MaltEval can be manipulated by flags, just as the evaluation arguments. The flags look similar to the evaluation flags. The value of the `argument` attribute of a `formatting` element is prefixed by two dashes (`--`). The set of formatting flags is thus `--micro-average`, `--header-info`, `--row-header`, `--details`, `--hdr-file`, `--stat`, `--tab`, `--output`, `--pattern`, `--confusion-matrix`. The `format` attribute then follows the formatting flag separated by a space character.

Note that only single values can be typed just as for the formatting arguments in the evaluation file. It is consequently not possible to specify multiple values separated by semicolons in a way similar to the evaluation flag values (see subsection 3.2). Note also that the order of any flag of any type of flag is irrelevant.

An example using all formatting flags is shown in appendix A as well as an equivalent call using an evaluation file instead.

5 Extending MaltEval using Java Plugins

MaltEval contains several type of grouping strategies that are useful for many users, but it cannot satisfy every possible need. It is therefore possible for users to write and compile their own pieces of Java-code implementing other grouping strategies, either completely new ones or by combining already existing grouping strategies with each other or new ones. They can then easily be integrated into MaltEval via plugins, without having access to the source code of MaltEval.

A new grouping strategy must implement the following Java interface:

```
package se.vxu.msi.malteval.grouping;
import java.util.Vector;
import se.vxu.msi.malteval.corpus.MaltSentence;
public interface Grouping {
  public void initialize();
  public Object getKey(MaltSentence sentence, int wordIndex);
  public boolean isComplexGroupItem();
  public DataContainer postProcess(DataContainer dataContainer);
  public boolean showTableHeader();
  public boolean showDetails();
  public String getSelfMetricName();
  public String correspondingMetric();
  public boolean isSimpleAccuracyGrouping();
}
```

The most important method is `getKey`, which take a sentence and an index of a word and returns a key. Have a look at the Javadoc for the interface above in the MaltEval distribution (located in the javadoc directory) for more information about what the methods above are used for. In addition, the distribution contains an example implementing `Grouping`. That example, with the name `ArcDirectionAndDeprel` is shown below as well:

```
import ...;
public class ArcDirectionAndDeprel implements Grouping {
  private Vector<String> validAttributes;
  private Vector<String> defaultAttributes;
  private ArcDirection arcDirection;
  private Deprel deprel;

  public Object getKey(MaltSentence sentence, int wordIndex) {
    return arcDirection.getKey(sentence, wordIndex) + " / " +
            deprel.getKey(sentence, wordIndex);
  }
  public Vector<String> getDefaultAttributes() {
    return defaultAttributes;
  }
  public Vector<String> getValidAttributes() {
    return validAttributes;
  }
  public void initialize() {
    arcDirection = new ArcDirection();
    deprel = new Deprel();
    arcDirection.initialize();
    deprel.initialize();
    validAttributes = MaltEvalConfig.getValidPrecisionAndRecallAttributes();
    defaultAttributes = MaltEvalConfig.getDefaultPrecisionAndRecallAttributes();
  }
  public boolean isComplexGroupItem() {
    return false;
  }
  public DataContainer postProcess(DataContainer arg0) {
    return arg0;
  }
  public boolean showDetails() {
    return true;
 }
  public boolean showTableHeader() {
    return false;
  }
  public String correspondingMetric() {
    return null;
 }
  public String getSelfMetricName() {
    return getClass().getSimpleName();
  }
  public boolean isSimpleAccuracyGrouping() {
    return false;
  }
  public String getSecondName() {
    return null;
  }
  public boolean isCorrect(int wordIndex,
   MaltSentence goldSentence, MaltSentence parsedSentence) {
  return getKey(goldSentence, wordIndex)
          .equals(getKey(parsedSentence, wordIndex));
  }
}
```

As the name of the class hints, this grouping strategy combines the grouping strategies `ArcDirection` and `Deprel`. The source code should be compiled together with the file `MaltEval.jar`, e.g.:

```
javac -classpath <path to MaltEval.jar> ArcDirectionAndDeprel.java
```

Place the created class file anywhere in a jar file, and copy it to a directory called *plugin* in the same directory as `MaltEval.jar`. When this is done, MaltEval is executed normally and the new grouping strategy is accessed in the same way as the default ones. MaltEval will search after all jar files in the plugin directory and dynamically load all class files in each jar file. MaltEval assumes that all class files in all jar files implement `Grouping`, and will otherwise terminate with an error. For example, applying the new grouping strategy is then done by simply typing:

```
java -jar MaltEval.jar --GroupBy ArcDirectionAndDeprel
   -s parsed.conll -g gold.conll
```

An easier way to compile new grouping strategies and create a jar file is to use the ant script file that comes with the MaltEval distribution (located in the directory plugin/ant). It automatically compiles and assembles grouping strategies located in a predefined Java source directory, but requires that ant is installed. See the README file in the plugin directory of the MaltEval distribution for information about installation of ant and execution of the ant script.

A Additional MaltEval Examples

This is a call to MaltEval having a large number of flags:

```
java -jar MaltEval.jar -s parser.conll -g gold.conll
   --Metric "LA" --GroupBy Deprel:all|recall-10
   --MinSentenceLength 1 --MaxSentenceLength 40
   --ExcludeDeprels ";Punc|Dummy" --details 1
   --header-info 1 --row-header 1 --tab 0
   --output result.out --pattern 0.0000 --stat 0
   --confusion-matrix 0
```

This is another call to MaltEval

```
java -jar MaltEval.jar -e eval.xml
   -s parser.conll -g gold.conll
```

which is equivalent to the call above if the file eval.xml contains the information below:

```
<evaluation>
   <parameter name="Metric">
      <value>LA</value>
   </parameter>
   <parameter name="GroupBy">
      <value format="all|recall-10">Deprel</value>
   </parameter>
   <parameter name="ExcludeDeprels">
      <value></value>
      <value>Punc|Dummy</value>
   </parameter>
   <parameter name="MinSentenceLength">
      <value>1</value>
   </parameter>
   <parameter name="MaxSentenceLength">
      <value>40</value>
   </parameter>
   <formatting argument="details" format="1"/>
   <formatting argument="header-info" format="1"/>
   <formatting argument="row-header" format="1"/>
   <formatting argument="tab" format="0"/>
   <formatting argument="output" format="result.out"/>
   <formatting argument="pattern" format="0.0000"/>
   <formatting argument="stat" format="0"/>
   <formatting argument="confusion-matrix" format="0"/>
</evaluation>
```

B File Formats

MaltEval is able to read a number of treebank formats. They are described below, with one subsection per file format.

B.1 CoNLL Format

The CoNLL data format adheres to the following rules:

- Data files contain sentences separated by a blank line.
- A sentence consists of one or tokens, each one starting on a new line.
- A token consists of ten fields described in the table below. Fields are separated by a single tab character. Space/blank characters are not allowed in within fields
- All data files will contain these ten fields, although only the ID, FORM, CPOSTAG, POSTAG, HEAD and DEPREL columns are guaranteed to contain non-dummy (i.e. non-underscore) values for all languages.
- Data files are UTF-8 encoded (Unicode).

A more detailed description is found here:
http://depparse.uvt.nl/depparse-wiki/DataFormat.

B.2 MaltXML Format

A dependency tree for the Swedish sentence “Genom skattereformen införs individuell beskattning (särbeskattning) av arbetsinkomster.” can be represented as follows:

```
<sentence id="2" user="" date="">
  <word id="1" form="Genom" lemma="genom"
    postag="pp" head="3" deprel="ADV"/>
  <word id="2" form="skattereformen" lemma="skattereform"
    postag="nn.utr.sin.def.nom" head="1" deprel="PR"/>
  <word id="3" form="införs" lemma="införa"
    postag="vb.prs.sfo" head="0" deprel="ROOT"/>
  <word id="4" form="individuell" lemma="individuell"
    postag="jj.pos.utr.sin.ind.nom" head="5" deprel="ATT"/>
  <word id="5" form="beskattning" lemma="beskattning"
    postag="nn.utr.sin.ind.nom" head="3" deprel="SUB"/>
  <word id="6" form="(" lemma="("
    postag="pad" head="5" deprel="IP"/>
  <word id="7" form="särbeskattning" lemma="särbeskattning"
    postag="nn.utr.sin.ind.nom" head="5" deprel="APP"/>
  <word id="8" form=")" lemma=")"
    postag="pad" head="5" deprel="IP"/>
  <word id="9" form="av" lemma="av"
    postag="pp" head="5" deprel="ATT"/>
  <word id="10" form="arbetsinkomster" lemma="arbetsinkomst"
    postag="nn.utr.plu.ind.nom" head="9" deprel="PR"/>
  <word id="11" form="." lemma="."
postag="mad" head="3" deprel="IP"/>
</sentence>
```

The tagsets used for parts-of-speech and dependency relations must be specified in the header of the XML document. An example document can be found here: http://w3.msi.vxu.se/∼nivre/research/example.xml.txt. It is worth mentioning that the word tag has the same number of attributes as the CoNLL format. They are named `id`, `form`, `lemma`, `cpostag`, `postag`, `feats`, `head`, `deprel`, `phead`, `pdeprel`, where `lemma`, `cpostag`, `feats`, `phead` and `pdeprel` are optional.

B.3 MaltTab Format

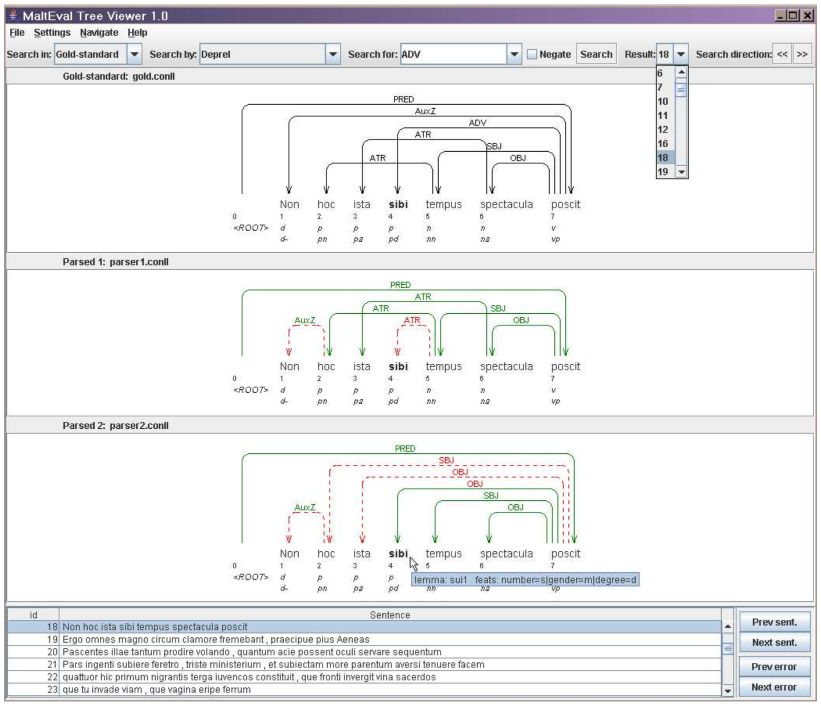
The corresponding sentence in MaltTab looks like this, which is a subset of the CoNLL data format:

```
Genom             pp                         3    ADV
skattereformen    nn.utr.sin.def.nom         1    PR
införs            vb.prs.sfo                 0    ROOT
individuell       jj.pos.utr.sin.ind.nom     5    ATT
beskattning       nn.utr.sin.ind.nom         3    SUB
(                 pad                        5    IP
särbeskattning    nn.utr.sin.ind.nom         5    APP
)                 pad                        5    IP
av                pp                         5    ATT
arbetsinkomster   nn.utr.plu.ind.nom         9    PR
.                 mad                        3    IP
```

Each row is divided into a number of fields separated by tab characters, no other white spaces, just as the CoNLL data format. The first column corresponds to wordsform, the second to postag, the third to head and the fourth to deprel. The example document can be found here:
http://w3.msi.vxu.se/∼nivre/research/example.tab.txt. Sentence splits in MaltTab are represented by blank lines as can be seen in the example document.

C MaltEval Tree Viewer: Example

The picture in 1 illustrates how a window containing a visual representation of three files (gold.conll, parsed1.conll, parsed2.conll) could look like. Green and red arcs and labels in the dependency trees of the two parsed files indicate whether the arcs and labels were correct or incorrect compared to the gold-standard.



Search in: Gold-standard Search by: Deprel Search for: ADV Negate Search Result: 18 Search direction: << >>

lemma: sui1 feats: number=s|gender=m|degree=d

id	Sentence
18	Non hoc ista sibi tempus spectacula poscit
19	Ergo omnes magno circum clamore fremebant , praecipue pius Aeneas
20	Pascentes illae tantum prodire volando , quantum acie possent oculi servare sequentum
21	Pars ingenti subiere feretro , triste ministerium , et subiectam more parentum aversi tenuere facem
22	quattuor hic primum nigrantis terga iuvencos constituit , que fronti invergit vina sacerdos
23	que tu invade viam , que vagina eripe ferrum

Figure 1: Example of the tree view in MaltEval

The bottom of the window shows a scroll list containing the sentences. By selecting another sentence, all subwindows above are updated so that the dependency trees for all files of that sentence are shown.