

RASP Evaluation Schemes

Rebecca Watson
Natural Language and Information Processing Group
University of Cambridge
Computer Laboratory
Rebecca.Watson@cl.cam.ac.uk

May 18, 2006

Abstract

This report outlines details of the relational dependency evaluation schemes written for RASP grammatical relations.

Contents

1	Introduction	2
2	Evaluation System	3
2.1	Output Formats	4
2.2	GR Match	5
3	Slot Comparison	8
3.1	All	8
3.2	Head-Dependent	8
3.3	Head-Dependent-NCSUBJ	8
4	GR-Type Match	8
4.1	Original	9
4.2	Equality Based	9
4.3	Subsumption Based	10
4.4	Hierarchy Based	10
5	Resources	11
5.1	Test Files and Gold Standard	11
5.2	Running the Evaluation System	12
6	Current System Performance	16

1 Introduction

The Robust Accurate Statistical Parsing (RASP) system for English developed by Briscoe & Carroll (2002) is publicly available for academic use. The parser is capable of returning a number of different output formats including grammatical relations (GRs). For details of the manually written tag sequence grammar and details of the GRs readers are referred to (Briscoe 2005).

This report contains details of the relational dependency based evaluation schemes enabling the reader to either a) replicate the schemes or b) interpret results output by the evaluation system. The test and gold standard files employed and the evaluation system outlined herein have been made publicly available for research to enable comparison of research efforts.

Briscoe (2005) outlines details of the GRs output by RASP. All GRs take the following form:

```
(GR-type subtype head dependent initial-GR)
```

The first item in this list: the GR-type, specifies the type of relationship between the head and dependent (e.g. (subj)ect, (obj)ect, (mod)ifier). The remaining items in the list: (subtype, head, dependent, initial-GR) will be referred to as ‘slots’ herein. The subtype and initial-GR are specified for only a subset of all possible GR-types. That is, their presence depends on the GR-type. In the case that these optional slots are present for a given GR-type the slot may appear with the default (unspecified) value of “_”.

For all evaluation schemes, both the GR-type and slot values (the value in the specified slot) must be ‘equal’. The set of evaluation schemes differ only in their definition of equality between GR-types and the set of slots to compare for slot ‘equality’. That is, they differ in the criteria for whether the test relation is correct: the test and standard items ‘match’. Section 2 outlines the overall architecture of the evaluation schemes while Section 4 and Section 3 outline alternative definitions of GR-type matching and the set of slots to compare, respectively.

We need to consider alternative sets of slots to compare as other parsers specify only head and dependent slots. Therefore, the evaluation system should be capable of comparing these slots only, to enable us to compare system performance.

There are 17 GR-types that form an inheritance hierarchy enabling relations to be underspecified. Readers are referred to (Briscoe 2005) for a figure of the hierarchy. This hierarchy of relations is shown in Figure 1. Note that the `passive` GR-type is not included in evaluation as this GR-type represents information available in the `ncsubj` GR-type when the `initial-GR` value is `obj`. This hierarchy enables a number of different schemes to be considered as we can define alternative definitions of GR-type match using this hierarchy. For example, we may consider GR-types in the test set (test, henceforth) that subsume GR-types in the gold standard set (standard, henceforth) as correct.

Section 6 reports current system performance using each of the evaluation schemes. Finally, Section 5 provides details of the files that are publicly available including the gold standard files and the file formats required to utilise the evaluation script.

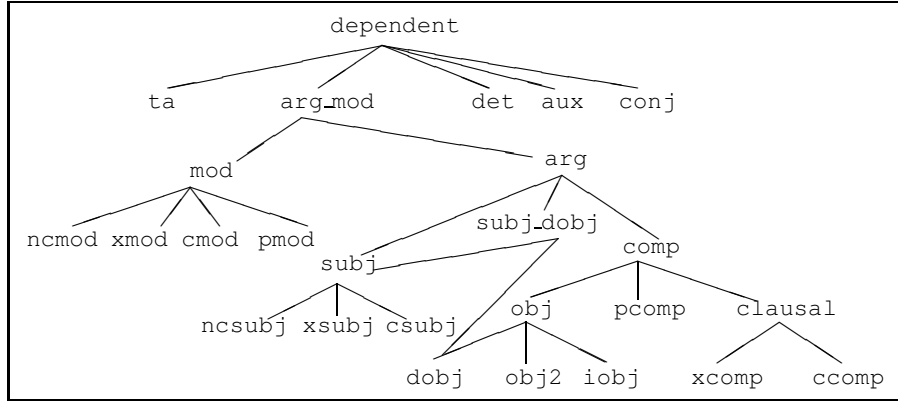


Figure 1: The GR hierarchy

2 Evaluation System

This section outlines the evaluation system’s architecture and highlights the functions which differ between each evaluation scheme (scheme, henceforth). These functions refer to the whether the GR-types match and which slots to compare (the definition of a slot match is constant between schemes). Once the reader is familiar with the general architecture, we will discuss the alternative sets of slots that may be compared and the GR-Type match options in Sections 3 and 4, respectively. Finally, Section 5 will outline the script file(s) that can be called to invoke the evaluation system over a set of test files and the script parameters that will select the scheme utilised by the system.

Each sentence is evaluated in-turn whereby the test and standard GR sets are compared. The test and standard files are required to contain the same number of sentences. An error message will be output by the system if this is not the case. The format required for test files is outlined in Section 5.

A GR in the test set ‘matches’ a GR in the standard set if the test GR is considered to be correct in the given scheme. Duplicate GRs are permitted in the standard set but not in the test set. If duplicate GRs occur in the test set (and not in the standard) then these are considered incorrect as only one match is permitted against each GR in the standard.

The evaluation system correlates for each sentence:

- **common:** the set of GRs that matched between the test and standard, where each element in this list are of the form: {standard GR, test GR}.
- **missing:** the set of GRs in the standard for which no test GRs were found to match.
- **extra:** the set of GRs in the test set for which no matches were found in the standard set.

After processing each sentence, the evaluation system outputs the list of common, missing, and extra GRs to the output file.

The evaluation system correlates across all sentences in the test set¹:

- **nsents**: the number of sentences in the test set.
- **std-total-GR-type**: the number of standard GRs of type *GR-type*.
- **tst-total-GR-type**: the number of test GRs of type *GR-type*.
- **agree-GR-type**: the number of GRs of type *GR-type* for which a match was found.

2.1 Output Formats

This section briefly outlines the alternative output formats available. The output file specified (see Section 5) contains for each sentence: the sentence number, the sentence, the list of common GRs (“In both”), the missing GRs (“Standard only”), and the extra GRs (“Test only”) outlined in the previous section. The sentence summary output format outlined in Section 2.1.3 is then output to the file. The micro- and macro-averaged scores for each relation (and for the test set overall as outlined in Section 2.1.2) are then output to the file (outlined in Section 2.1.1).

2.1.1 Relation Summary

The relation summary format reports the precision, recall, and F_1 for each GR-type. For all but one option these are computed by percolating up the counts for std-total- tst-total- and agree- up the hierarchy, reflecting the average performance across each GR-type and it’s associated sub-types in the hierarchy. However, for the hierarchy method outlined in Section 4.4, these counts are not percolated upwards in a routine fashion. Section 4.4 outlines full details of how the counts are percolated upwards.

Once the counts have been percolated upwards in the hierarchy we can determine precision, recall, and F_1 for each GR-type using:

$$precision = \frac{agree - GR - type}{tst - total - GR - type} \quad (1)$$

$$recall = \frac{agree - GR - type}{std - total - GR - type} \quad (2)$$

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \quad (3)$$

Note that the relation summary includes the summary output outlined in Section 2.1.2.

¹Note that the std-total- tst-total- and agree- scores are recorded for each GR-type

2.1.2 Summary

The summary format reports the micro-averaged and macro-averaged precision, recall, and F_1 scores over all relations. The micro-averaged scores represent the performance of these performance measures considered over all GR-types. Thus, these scores are identical to the scores reported for the `dependency` relation in the relation summary format outlined in the previous section.

The macro-averaged scores represent the average of each score over the 17 GR-types. However, these scores are not calculated using the percolated counts. Precision, recall and F_1 are determined for each GR-type using the raw counts of `std-total`, `tst-total`, and `agree` over each GR-type. The average of each of these measures over each GR-type is then calculated to report the ‘macro’ average of the scores. These scores place equal importance over each GR-type rather than over those that occur more frequently.

2.1.3 Sentence Summary

This format outputs precision, recall and F_1 for each sentence in the test set. These scores are determined for each sentence using the equations outlined in Section 2.1.1.

2.2 GR Match

This section outlines the definition of a ‘match’ between two GRs (test and standard). This definition holds over all GR-type definitions except for the hierarchy method outlined in Section 4.4, which describes slight modifications to this definition.

In order for the test GR (tst) to match a given standard GR (std) the following must hold:

- the GR-types of std and tst match and
- the slots (the values in the slots) match.

As previously mentioned, the definition of a GR-type match differs between schemes and the set of slots to compare differs. Section 2.2.1 will define the slots that occur in each GR-type and Section 2.2.2 gives the definition of a match between two given slots.

2.2.1 Slots

Figure 2 illustrates the list of 16 possible GR-types (we do not include the `passive` GR-type in evaluation) and the set of associated slots. For each GR-type all and only these slots must occur. Though the slot values may contain “_” if the optional slot is unspecified. The system will output an error message if a GR-type does not occur with the specified number of slots shown in the figure.

```

(dependent subtype head dependent)
(mod        subtype head dependent)
(ncmod      subtype head dependent)
(xmod       subtype head dependent)
(cmod       subtype head dependent)
(pmod       head dependent)
(det        head dependent)
(arg_mod    subtype head dependent)
(arg        subtype head dependent)
(subj       head dependent initial-gr)
(ncsubj     head dependent initial-gr)
(xsubj      head dependent initial-gr)
(csubj      head dependent initial-gr)
(subj_dobj  head dependent)
(comp       head dependent)
(obj        head dependent)
(dobj       head dependent)
(obj2       head dependent)
(iobj       head dependent)
(clausal    head dependent)
(xcomp      subtype head dependent)
(ccomp      subtype head dependent)
(pcomp      head dependent)
(aux        head dependent)
(conj       head dependent)
(ta         subtype head dependent)
%(passive   head)

```

Figure 2: The GR-types with associated required slots.

2.2.2 Slot Match

A number of options are available for the evaluation system. One option outlines which set of slots to compare. However, the definition of a slot match remains constant across these options. Two slots (of the same type) are considered to match if one of the following constraints hold given the slot values `tst-arg` and `std-arg` for test and standard slots, respectively:

1. the values `tst-arg` and `std-arg` are identical (including unspecified value of `_`).
2. `tst-arg` and `std-arg` are both specified (not `_`) and:
 - (a) either `tst-arg` or `std-arg` specifies the value `ellip`, signifying that that the argument references an ellipsis or
 - (b) `std-arg` is a word in the multiword `tst-arg`. A multiword represents an NE in the text where words are separated by `_`.
3. either `tst-arg` or `std-arg` are unspecified (`_`) and:
 - (a) the slot is of type `type` and
 - (b) the GR-type is one of `mod`, `ncmod`, `xmod`, `cmod`, `pmod`, `arg`, `xcomp`, `ccomp`, `ta`.

Note that the slot match criteria 2b accounts for the use of named-entity recognition during parsing. That is, the test files include versions of the text and gold standard using NE markup. In this case the last word in the NE item appears in the gold standard. When parsing RASP will output these multiword constituents as a single word where words are separated by `"_"`. For example, the second sentence in the test file is²:

^ The following issues were recently filed with the
<w>Securities and Exchange Commission</w>:

In this case, the test file (the `.parses` file outlined in Section 5) should include GRs containing the multi-word item `'Securities_and_Exchange_Commission'` to enable the slot value to match with the standard GRs for the sentence:

```
(ncsubj filed issues obj)
(ncmod _ filed recently)
(aux filed were)
(passive filed)
(iobj filed with)
(dobj with Commission)
(det Commission the)
(det issues The)
(ncmod _ issues following)
```

²The beginning of sentence marker is `^`.

3 Slot Comparison

The definition of a slot match was given in Section 2.2.2. In this section we will outline the alternative slot comparison options available. That is, options that define which set of slots to compare. All slots in the set compared must match for the slots to match.

These options, outlined in the following sections, can be summarised as follows:

- All: all slots are compared.
- Head-Dependent: only head and dependent slots are compared.
- Head-Dependent-NCSUBJ: in addition to comparing the head and dependent slots, we compare the `initial-GR` slot for the `ncsubj` GR-type if the value of the slot is `obj` in either the test or standard.

3.1 All

For the ALL slot comparison option, all slots are compared between the standard and test relation. For each slot in the standard relation, we determine whether or not this slot is present in the test relation. If the slots are both present in the test relation then we compare the slots. If a slot is not present in either the test or standard relation then the slot is considered correct by default. All slots that are present in both the standard and test relation are required to match.

For example, if the standard relation and the test relation specify `ncmod` and `iobj`, respectively, then we will compare the slots `head` and `dependent` only and we will ignore the `type` slot. If the values in the `head` and `dependent` slots match then the slots are considered to match.

Note that the original evaluation scheme uses a modified version of this definition: the number of slots in both the standard and test relations must be equal. However, the scheme does not specify that these slots must contain the same set of slot types. We relaxed this original definition for the new schemes (as outlined above) to enable comparison between the slots of any two GR-types.

3.2 Head-Dependent

For the HEAD-DEPENDENT option, only the head and dependent slots are compared.

3.3 Head-Dependent-NCSUBJ

For the HEAD-DEPENDENT-NCSUBJ option, the `head` and `dependent` slots are compared. Further, if the GR-type is `ncsubj` then we compare the `initial-GR` slot if the slot value is `obj` in either the test or standard.

4 GR-Type Match

This section outlines the various options for determining a GR-type match. That is, whether or not the GR-type of the standard relation (`std-GR`) and test relation (`tst-`

GR) match. These options, outlined in the following sections, can be summarised as follows:

- Original: the tst-GR can be
 - equal to the std-GR,
 - more general than the std-GR by one level in the hierarchy or
 - more specific than the std-GR by any number of levels in the hierarchy.
- Equality: the tst-GR must be equal to the std-GR.
- Subsumption: the tst-GR can be:
 - equal to or
 - more general than the std-GR by any number of levels in the hierarchy.
- Hierarchy: this scheme returns the most specific GR-type in the hierarchy that subsumes both the tst-GR and std-GR. A match is always returned and, thus, a match is considered to occur at this level in the hierarchy.

4.1 Original

The original GR-type match can be summarised as follows, the tst-GR can be:

- equal to the std-GR,
- more general than the std-GR by one level in the hierarchy or
- more specific than the std-GR by any number of levels in the hierarchy.

The original scheme enables the tst-GR to be more general by one level in the hierarchy, and only if the std-GR is a leaf node in the hierarchy. Note that the current standard does not contain any GR-types that are not leaf nodes in the hierarchy. Therefore, the test GR-type will never be more specific than the standard.

4.2 Equality Based

The original GR-type match scheme outlined in the previous section enables the GR-type to be more general than the standard. Therefore, the reported performance does not provide information on how well the parser is determining a particular GR-type. For example, we may wish to know how well the system performs in terms of the `ncmod` GR-type. Currently, if the standard contains a `ncmod`, the test may return a `mod` or an `ncmod`. Thus the user does not know the individual counts of these two scenarios. Two systems will seem to perform equally as well even if one returns the more precise match `ncmod` more frequently than the other. Therefore, the equality based GR-type match measures the exact performance of each GR-type. That is, it performs GR-type matching by requiring the GR-type to be equal.

4.3 Subsumption Based

As previously mentioned, the GR-types present in the standard are only those that occur as leaf nodes in the hierarchy. That is, the standard contains only the most specific GR-type possible. Thus, the test relation will never be more specific than the standard and it would be more beneficial to determine performance by enabling test relations to be more general by *any degree* in the hierarchy. Therefore, the subsumption based GR-type match considers the tst-GR and std-GR to match if tst-GR is equal to or more general than the std-GR.

4.4 Hierarchy Based

While the previous schemes enable the tst-GR to be either equal to and optionally more general than the std-GR, they do not report the unlabelled dependency accuracy. The unlabelled dependency performance reflects how well we determine any type of dependency relation. That is, how well we perform just on matching the required slots. Many parsers report this measure and it would be beneficial for comparison if the evaluation system reported this performance.

The unlabelled dependency performance reflects how well we do at the level of `dependent` in the hierarchy. That is, if we enable any two GR-types to match and define a GR match to occur if the required slots all match. Similarly, we may wish to determine how well we do at the level of `mod`. At `mod` level of the hierarchy we can define that two GR-types match if they are both one of: `mod`, `ncmod`, `cmod`, `pmod`. Determining a score for `mod` in this way enables us to compare the performance of RASP at determining modifiers to another parser that does not make the fine-grained distinction between modifier types. That is, we do not penalise RASP for trying to make finer-grained distinctions.

Thus, at any level in the hierarchy we wish to see how well we do if we enable GR-type match to occur between any two GR-types at that level or lower in the hierarchy. We determine the ‘GR-type match’ as the the most specific GR-type in the hierarchy that subsumes both the tst-GR and std-GR GR-types. A GR-type match is always returned for a given test GR (tst-GR) and list of standard GRs (std-GR-list). As previously mentioned, a GR match is determined between a test and standard GR if the GR-types match and the set of required slots match. However, we do not wish to match to the first item in the std-GR-list for which the slots match. Instead, we wish to match the tst-GR to a standard GR for which the slots match and also the GR match (the GR-type that subsumes both the test and standard GR) is the most specific.

Given that the relation `tst-GR` finds a matching std-GR (the slots match) with the GR-type match *grtype-common*, we percolate up counts for `std-total`- `tst-total`- and `agree`- GR-types as follows:

- *std-total-GR-type*: increment this count for the given GR-type in the hierarchy if the *GR-type* is equal to or subsumes the std-GR GR-type.
- *tst-total-GR-type*: increment this count for the given GR-type in the hierarchy if the *GR-type* is equal to or subsumes the tst-GR GR-type.

- *agree-GR-type*: increment this count for the given GR-type in the hierarchy if the *GR-type* is equal to or subsumes the *grtype-common*.

Note that for each GR in ‘missing’ or ‘extra’ lists outlined in Section 2 we also percolate up the counts for *std-total-* and *tst-total-*, respectively, for each GR-type as above.

5 Resources

This section outlines all of the key files which comprise or are required by the evaluation system. Section 5.1 gives details of the gold standard files and required format of the test files. These resources can be found in `$RASP/extra/neweval` (`$RASP-EVAL`, henceforth). Example files have been included in the `$RASP-EVAL` directory: `example.out`, `example.grtext`, `example.parses` and `example.output` as well as the NE versions of each.

5.1 Test Files and Gold Standard

5.1.1 Gold Standard - Parc700 Depbank

King, Crouch, Riezler, Dalrymple & Kaplan (2003) outline the development of the PARC 700 Dependency Bank (henceforth, DepBank), a gold-standard set of relational dependencies for 700 sentences (originally from the Wall Street Journal) drawn at random from Section 23 of the Penn Treebank. Briscoe & Carroll (2005) extend DepBank with a set of gold-standard GRs and (manually corrected) PoS tags. The extended DepBank file can be found at `$RASP-EVAL/gold700files.rasp`, note that this file includes all 700 sentences.

We will use the gold-standard GRs to measure parser accuracy, over the same 560 sentence test suite from the DepBank utilised by Kaplan, Riezler, King, Maxwell, Vasserman & Crouch (2004). The 560 test suite subset and the NE version of this subset can be found in `$RASP-EVAL/parc700/gold560.rasp` and in `$RASP-EVAL/parc700/gold560ne.rasp`, respectively.

5.1.2 Text and Pre-processed Files

The text file (to parse) and the NE version of this text file can be found in `$RASP-EVAL/parc700/test.not-ne` and in `$RASP-EVAL/parc700/test.ne`, respectively. These files have sentence boundaries automatically detected and the `-s` option should be used when invoking `$RASP/scripts/rasp.sh` over these files.

Alternatively, the user can employ `$RASP/scripts/rasp_parse.sh` using the pre-processed versions of `test.not-ne` and `test.ne`: `$RASP-EVAL/parc700/test.not-ne.stag.data` and `$RASP-EVAL/parc700/test.ne.stag.data`, respectively. Note that these files contain the pre-processed text using the PoS tagger in forced-choice mode.

5.2 Running the Evaluation System

Two scripts are provided: `eval.sh` and `eval_system.sh`. This section outlines the details of utilising these script files that invoke in the evaluation system. The `eval.sh` script is designed to pre-process the RASP output file ready for the `eval_system.sh` script. The resulting file formats prepared by `eval.sh` are outlined in Section 5.2.1. The `eval_system.sh` script accepts these pre-processed input files and a number of input parameters that define the evaluation options including the set of slots to compare and the definition of GR-type match as outlined previously.

The input parameters for each script are described in Section 5.2.2. Example invocation (the same evaluation scheme is applied in each) over parser out file `example.out`:

```
./eval.sh -d -t example -e'-u -o -s n -g s -m'  
./eval_system.sh -t example -c sents-tst.gr -u -o -s n -g s -m
```

5.2.1 Input Formats

`$RASP-EVAL/eval.sh` takes the RASP output format in file `x.out` (see e.g. `$RASP-EVAL/example.out`) and produces `x.trans.grtext` and `x.trans.parses` (or `x.grtext` and `x.parses` depending on which evaluation options are utilised) in the format required by `$RASP-EVAL/eval_system.sh`. These files contain the sentences and GR sets from `x.out`, respectively.

This section outlines the required formats for `.grtext` and `.parses` files to enable users to utilise the evaluation system with other parsers, if required. The next section outlines the input parameters available to each script.

Users who employ the RASP parser can utilise the `eval.sh` script to automatically create these input files from RASP output file `x.out`. `Eval.sh` then invokes the `eval_system.sh` script with these input files.

Note that the files input into `eval_system.sh` must contain GRs in which the original word occurs and not the morphological variant of the word e.g. `$RASP-EVAL/parc700/test.not-ne.stag` compared to `$RASP-EVAL/parc700/test.not-ne.stag.data`. The `eval.sh` script automatically converts the GR words from e.g. `asset+s` to `assets` using the `eval_transform.sh` script. If the user wishes to parse the `$RASP-EVAL/parc700/test.(not-)ne.stag` file instead which does not include morphological variants (so that RASP will output GRs with the original word forms) then the user should use the `-x` option for `eval.sh` outlined below. However, performance using subcategorisation or phrasal verbs (using the `rasp_parse.sh` options `-s` and `-x` respectively) will not correspond to the true performance as the stem of the verb is required by RASP in these cases. Therefore, the user is directed to parse the `$RASP-EVAL/parc700/test.(not-)ne.stag.data` files which include the morphological variants of words.

- `x.grtext`

An example of the format required is shown in `$RASP-EVAL/example.grtext`. The file is required to have the sentence number (starting at 1) followed on the next line by the sentence itself. The sentence should be tokenised into the set that specifies the possible items that may appear in the GR slots as values. The

evaluation system will output an error if a value occurs in a GR slot that is not present in this sentence. A blank line should follow each sentence.

- **x.parses**

An example of the format required is shown in \$RASP-EVAL/example.parses. The example file begins with two lines that can be ignored and these lines do not necessarily have to occur in the .parses file. All input prior to the first sentence (the token 1 shown on line 4 of example.parses) will be ignored. Each sentence should be numbered (starting from 1) followed by a blank line and then the GR set for the sentence (where one GR occurs per line). A blank line should follow each GR set.

Both the x.parses and x.grtext contain information for each sentence, where the information is labelled by the sentence number. Note that each file input to eval_system.sh should specify the same number of sentences otherwise an error will be returned by the system. Further, the evaluation system ensures that the same sentence number is compared from x.grtext, x.parses and the standard file. Therefore, these files must contain the sentence information labelled with the correct sentence number (sentence numbering starts at 1 for all files).

5.2.2 Script Parameters

Both eval.sh and eval_system.sh can be used to invoke the evaluation system (in eval_system.sh). The eval.sh script should be used if the user first wishes to create the input files x.grtext and x.parses from the RASP output file x.out. The switches all have a default value of false. Example invocations (calling equivalent schemes in the evaluation system):

```
./eval.sh -d -t example -e' -u -o -s n -g s -m'
./eval_system.sh -t example -c sents-tst.gr -u -o -s n -g s -m
```

Eval.sh parameters:

- **-t <test-file>**

This option is used to specify the test file name (test-file). That is, the x in the RASP output file x.out. Note that if the -r switch is used the test file specified will be created by parsing the appropriate test suite.

- **-r : run rasp switch**

This switch specifies to run rasp over the text file \$RASP-EVAL/parc700/test.note to create the specified test-file.out file.

- **-n : NE switch**

This switch specifies that sentences with NE mark-up were or are to be parsed. Therefore, the file sent-ne-tst.gr will be used as the standard in place of the sents-tst.gr. Note that these standard files are automatically created by eval.sh using either \$RASP-EVAL/parc700/gold560.rasp or \$RASP-EVAL/parc700/gold560ne.rasp.

Therefore, the most recent version of the gold standard(s) will be employed during evaluation.

Used in conjunction with the -r switch results in RASP parsing the text file with NE mark-up \$RASP-EVAL/parc700/test.ne instead of \$RASP-EVAL/parc700/test.not-ne.

- **-x : translate switch**

This switch specifies that test-file.out file contains GRs that specify the original word already (in the case that the user has deliberately parsed \$RASP-EVAL/parc700/test.(not-)ne.stag file instead of \$RASP-EVAL/parc700/test.(not-)ne.stag.data using the rasp_parse.sh script. Hence this switch specifies that the eval_transform.sh script should not be applied.

- **-d : debug switch**

This switch should be used if the user wishes to keep copies of the x.grtext and x.parses files. Otherwise these files are removed after the evaluation system terminates.

- **-e <eval_system parameters>**

The -e option specifies (in single quotes, e.g., -e'-o -u') the evaluation system parameters to be passed to the eval_system.sh script. These parameters are outlined below.

Eval_system.sh parameters:

- **-t <test-file>**

This option is used to specify the test file name (test-file). That is, the x in the RASP output file x.out and the input files x.grtext and x.parses.

- **-c <std-file>**

This is the name of the standard file: sents-tst.gr or sents-ne-tst.gr depending on whether NE markup was used.

- **-g <gr-type match>**

The different matching schemes for GR-types is outlined in Section 4 including (e)quality-, (s)ubsumption- and (h)ierarchy-based matching. Hence, the specified parameters passed with the -g switch are e, s, and h to select these schemes, respectively. The default value is h.

- **-s <slot-compare>**

Section 3 outlines the various sets of slots that may be compared including (a)ll, (h)ead-dependent, head-dependent-(n)csubj. Hence, the specified parameters passed with the -s switch are a, h, and n to select these schemes, respectively. The default value is a.

- **-o : output the original evaluation schemes' performance**

This switch enables the user to output the original evaluation schemes' performance in addition to the new schemes'. Note that feedback on which GRs matched is provided for each sentence as well (i.e. common, missing and extra GR lists outlined in Section 2).

- **-u : output the unlabelled dependency performance**

This switch enables the user to output the micro- and macro-averaged scores for unlabelled dependency performance in addition to the new schemes'. Again, feedback on which GRs matched is provided for each sentence as well.

- **-m : output the confusion matrix to standard-out**

This switch enables the user to output the confusion matrix for the new scheme specified.

5.2.3 Error Messages

This section briefly outlines the possible error messages (output by eval_system.sh) and why these errors occur:

- **“Reached end of file at different times”**

The user has input .grtext .parses and a standard file in which the number of sentences differs between files. That is, one or more files does not contain enough sentences compared to the longest file. The .grtext and/or .parses files must be modified so that the same number of sentences occurs in all three files.

- **“Mismatch in sentence numbers: W (text), S (standard), T (test)”**

The user has input .grtext .parses and a standard file in which the sentences numbers differ at a point in the files. The files are expected to contain sentence numbers starting from 1 and incrementing by 1 until the end of file. If a sentence number is missing in one of the input files then this error will occur. The user is provided with the sentence number specified in the text, standard and test files: W, S, and T, respectively. The .grtext and/or .parses files must be modified so that the same sentence numbers occur in order in all three files.

- **“Unexpected end of file, stream S”**

The stream S may be one of text-str, std-str, or tst-str for the .grtext .parses and standard file, respectively. The stream specified ended while a GR was being read in. The user should check the respective file's end to correct the source of error.

- **“Expecting left parenthesis at byte B, stream S”**

The stream S may be one of text-str, std-str, or tst-str for the .grtext .parses and standard file, respectively. The system was expecting to read in '(' (the beginning of a GR) at byte B but found a different character. The user should check the respective file's end to correct the source of error.

GR-type/Model	All		Head-Dependent-NCSUBJ		Head-Dependent	
	Micro-F ₁	Macro-F ₁	Micro-F ₁	Macro-F ₁	Micro-F ₁	Macro-F ₁
Original	71.73	61.25	-	-	-	-
Equality	71.24	56.32	71.49	56.52	71.49	56.52
Subsumption	71.53	59.60	71.78	59.79	71.78	59.79
Hierarchy	73.68	61.98	73.94	62.21	73.94	62.21
Unlabelled	75.91	75.91	76.16	76.16	76.16	76.16

Table 1: Parser Performance over Parc DepBank using different Slot selection schemes and GR-type match schemes.

- **“Unknown relation name G”**

The relation name G is not in the set of possible GR-types allowed (see Figure 2 containing the list of allowable GR-types). This could occur in any of the input files.

- **“Not enough arguments to relation in G - expecting S”**

An input file contains a GR G that does not contain the specified number of slots. The slots expected are reported as S.

- **“Wrong number of slots in relation G - expecting S”**

This error message specifies the same error as the previous message.

6 Current System Performance

This section briefly outlines the ‘current’ performance³ of the RASP system (without NE markup) measured using the range of evaluation schemes available. The performance can be summarised in Table 1, while system performance using NE mark-up is shown in Table 2.⁴

References

- Briscoe, T. (2005), An introduction to tag sequence grammars and the rasp system parser.
- Briscoe, T. & Carroll, J. (2002), Robust accurate statistical annotation of general text, *in* ‘Proceedings of the Conference on Language Resources and Evaluation (LREC 2002)’, Palmas, Canary Islands, pp. 1499–1504.

³This performance was measured in December 2005 since which time the system has been modified. Users can ascertain the performance of the system they currently utilise via the scripts outlined herein.

⁴Performance for the schemes was derived by pre-processing the input. Given (*ncsubj* Vb x i) and (*passive* Vb) GRs where Vb is a given verb and x and i can hold any value: we remove the (*passive* Vb) GR from the GR set and ensure that i is set to the value *obj*.

GR-type/Model	All		Head-Dependent-NCSUBJ		Head-Dependent	
	Micro-F ₁	Macro-F ₁	Micro-F ₁	Macro-F ₁	Micro-F ₁	Macro-F ₁
Original	72.96	61.65	-	-	-	-
Equality	72.47	56.75	72.74	56.96	72.74	56.96
Subsumption	72.78	59.97	73.05	60.19	73.05	60.19
Hierarchy	74.84	62.42	75.11	62.66	75.11	62.66
Unlabelled	77.56	77.56	77.83	77.83	77.83	77.83

Table 2: Parser Performance over Parc DepBank NE data using different Slot selection schemes and GR-type match schemes.

Briscoe, T. & Carroll, J. (2005), Evaluating the speed and accuracy of a domain-independent statistical parser on the PARC Depbank. Submitted.

Kaplan, R., Riezler, S., King, T., Maxwell, J., Vasserman, A. & Crouch, R. (2004), Speed and accuracy in shallow and deep stochastic parsing, *in* ‘Proceedings of the Human Language Technology conference / North American chapter of the Association for Computational Linguistics annual meeting’, Boston, Massachusetts, pp. 97–113.

King, T., Crouch, R., Riezler, S., Dalrymple, M. & Kaplan, R. (2003), The PARC700 Dependency Bank, *in* ‘Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)’.