From Corpus to Codings: Semi-Automating the Acquisition of Linguistic Features

Michael O'Donnell
Department of AI, University of Edinburgh,
80 South Bridge, Edinburgh. EH1 1HN, UK.
email: micko@aisb.ed.ac.uk

February 5, 1996

Abstract

This paper describes a tool that facilitates the linguistic coding of corpus material, through the efficient prompting of the user for relevant categories. Linguistic features are organised in terms of an inheritance network to reduce the amount of coding effort. A *corpus-exploration tool* allows the user to view only those codings matching particular criteria. The tool also allows some forms of statistical analysis, particularly comparisons between specified subsets of the corpus. Alternatively, codings can be exported in a form readable by statistical packages.

1 Introduction

To perform text studies, we often need to spend significant amounts of time coding our texts – splitting them up into segments of some size, and assigning features (discourse, syntactic, etc.) to each segment. We then have the problem of re-representing the coded information in a format which can be used for statistical analysis.

Ideally, some form of automatic coding of the text will be performed, using a tagger, syntactic parser, or semantic analyser. Unfortunately, the scope of such tools is limited (both in terms of syntactic coverage and semantic depth), particularly when discoursal features are being coded.

The alternative to fully automatic coding is semi-automated coding. Over the last few years, I have been developing a software tool to semi-automating some of the processes involved in coding text. The result of this work is called the "WAG Coder", which is one module of the Workbench for Analysis and generation (WAG) system – a system for single-sentence analysis and generation (O'Donnell 1994, 1995b). The program runs on Macintosh computers.

The WAG Coder uses a menu-driven, window-based interface to maximally simplify the coding task. The user is prompted with a series of linguistic alternatives (choices) from which the user chooses one. Further choices are then presented.

The coder can be set up to code text units at any linguistic level, for instance, graphological status, discoursal features, or sociological variables. However, the user does need to provide the coding scheme, which is a statement of the features to be coded, also stating which of these features are mutually exclusive. The systemic term for a set of mutually exclusive features is a *system*.

It is useful to avoid coding choices which do not apply to the present unit. For instance, if we are coding an intransitive clause, it doesn't make sense to ask whether the clause is active or passive. By using a *systemic network* (systems organised into an inheritance network) to represent the relations between features, we avoid this problem. Some choice alternatives (systems) are made dependent on prior features being chosen. Choice sets are thus ordered in dependency.

The WAG Coder was developed under the Electronic Discourse Analyser project (Matthiessen *et al.* 1991), funded by Fujitsu (Japan). Faced with the need for grammatical profiles of our target texts, and lacking analysis tools, we developed the coder to help us build the profile. The Coder was further developed under an NSF-funded project (National Science Foundation Grant IRI-9003087) to study the register of Newspaper articles, as part of a wider goal of making the output of a text generation system sensitive to register variation (see Bateman & Paris 1989a, 1989b; Paris & Bateman 1990).

A number of other semi-automatic coding tools are available. Bliss & Ogborn (1983) discuss one coder, also using system networks. However, this is a relatively dated program using a text interface. Webster (199x) discusses another coder, also for Macintoshes, which allows the user to assign function structure to text. Alexis & ?? discuss another system which [to complete].

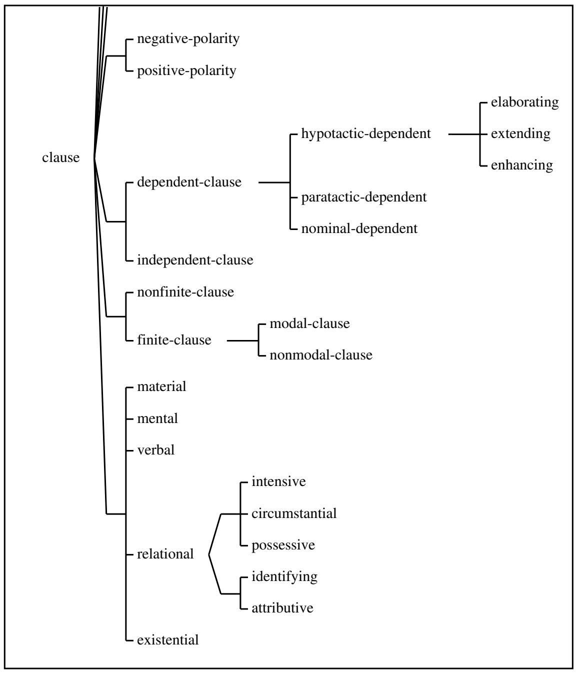

Figure 1: A Partial Graph of a Coding Scheme

2 Pre-Preparation

2.1 The Corpus

To prepare the corpus, the user needs to pre-segment the text, one item per line of a text file, e.g., for a study which is studying the expression of semantic events:

```
Creating a DASD dataset
This section describes the knowledge required
to create a DASD dataset.
A DASD dataset can be created
by specifying NEW in the DISP parameter
     of a DD statement.
Alternatively, the DASD dataset can be created
etc.
```

2.2 The Coding Scheme

The user must represent the coding scheme (the features in which the user is interested) in terms of a system network. This network needs to be entered into the computer in the format which is used for entering grammars in the WAG system. The input format is similar to that used in the Penman Text Generation system (WAG does in fact read Penman-format systems):

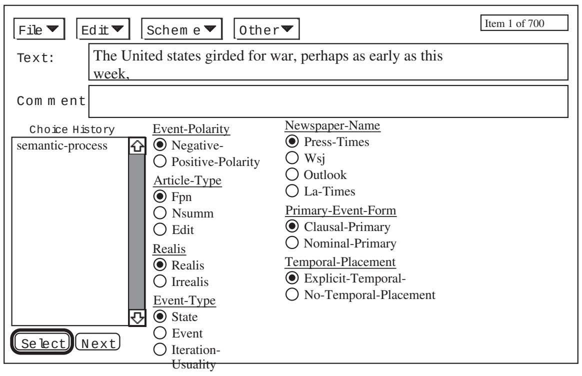

Figure 2: The Coder Window

```
(defsystem
  :name congruency
  :entry-condition semantic-event
  :features (clausal-event
             nominalised-event
             adjectival-event))
```

The user provides a set of these systems, which together define a system network. These are read into the coder, which can then be used for semi-automated coding of the text corpus using this coding scheme.

The features in the coding scheme can be from any linguistic level, for instance, intonational, grammatical, semantic, speech-function, contextual (e.g., the gender of the speaker, the source of the text). These levels may be mixed freely within the coding scheme.

The user can use the Systemic Grapher, another module of the WAG system, to check that the coding scheme has been defined as intended. Figure ?? shows a part of a graph of a typical coding scheme.

3 Feature Coding

Once the text has been prepared, and the coding scheme entered, the user selects “Code Text” from a menu. A dialog window appears, with several boxes (see figure ??). The user then nominates which text file should be loaded, containing the instances to code. The interface will then present the user with each coding instance in turn (each line of text from the text file) and prompt the user to choose features for each item.

3.1 Feature selection

At the left of the Coding window is a scrolling dialog, labelled “Choice History” which shows the features you have selected so far for this item (initially showing just the start feature)

Rather than stepping through each system in the coding network, the user can be presented with a dialogue window displaying all systems which are currently relevant (the condition on the system has been satisfied). See figure ??. One feature in each system is marked as the default. The user can change the default selection by clicking on one of the non-default option. When the appropriate features are selected in each system, the user presses the "Select" button, and the choices are recorded. This approach allows a large number of features to be coded with minimum effort, especially where most instances conform to the default coding.

The second of these is labelled "Select Feature". This displays the first system in the network. If you double click on one of these choices, the feature is selected and moved to the other list the "Choice History" box. The Coder will then find the next system to the left in the system network, and present them with the choices.

In this manner, the system network is automatically traversed, the Coder prompting the user at each point. All of this proceeds in a quick and easy manner, allowing substantial amounts of instances to be coded quite quickly.

When no further choices remain, the user presses the "Store" button, which saves this coding away to a designated file. Codings can be re-accessed later for re-editing if desired.

3.2 Changing Your Mind: Deleting Features

To delete features from the "Choice History", just double-click on the relevant feature. The feature, and all the features which depend on the choice, will be removed from the Choice History.

4 Post-Editing of Codings

Various tools exist to view and edit codings once they have been made.

4.1 Editing Codings

The interface allows the user to call up any stored codings, and change the feature codings, comments, or text-string. From the Coder interface, you press the "View/Edit" button, and a list of all codings appears (see figure ??). Double-click on any coding, and an editor will appear. This interface also allows you to delete codings.

4.2 Filtering Codings

The "View/Edit" interface also allows you to view codings which fit a particular feature specification. Type in a feature specification (either a feature, or a logical combination of features), and only those codings which match the feature-specification will be displayed. For instance, using my coding network, I can type in any of the following feature specifications:

- **material**: Shows all material clauses in the corpus.
- **(and material abstract)**: Shows all material clauses in the Abstract stage of the text.
- **(not material)**: Shows all clauses which are not coded as material.

Feature-specifications can be arbitrarily complex, e.g. (or (not active) past). Once the feature specification is typed in, press the "Apply" button, and the restricted set of codings will be shown. If you leave the feature-specification field blank when you press the "Apply" button, then you will be presented with a list of all features. Choose one to use as the filter.

4.3 Updating Codings

If you need to change the coding scheme at any point, either changing the inheritance of categories, adding features, or adding whole systems, then the Coder allows you to update past codings without re-coding the information you already have. In the "Update Codings" mode, the coder loads up a file of saved codings, and checks the stored features against the present coding scheme. The coder will then prompt only for systems which it has no recorded feature.

5 Corpus Exploration

Once coded, the codings represents a tagged corpus – each text item is tagged with a set of features. We may then wish to explore this corpus, selecting out instances which conform to some feature specification.

The Systemic Coder includes a tool which facilitates the browsing through the corpus. This is the Review Window introduced above.

Part of this interface, not so far discussed, is its filtering capability – it lets you view only those codings which fit a particular feature specification. Type in a feature specification (either a feature, or a logical combination of features), into the filter box at the bottom of the screen and press the "Apply" button, and the display will change to display only those codings which match the feature-specification.

For instance, using my coding network, I can type in any of the following feature specifications:

- material: Shows all material clauses in the corpus.
- (and material abstract): Shows all material clauses in the Abstract stage of the text.
- (not material): Shows all clauses which are not coded as material.

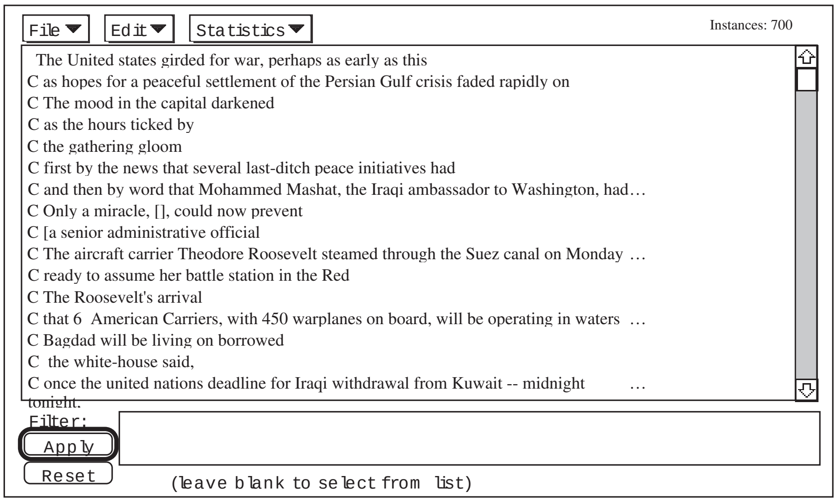

Figure 3: The Review/Edit Window

Feature-specifications can be arbitrarily complex, e.g. (or (not active) past). Once the feature specification is typed in, press the "Apply" button, and the restricted set of codings will be shown.

Editing, deleting and inserting will apply as before, even in the filtered mode.

Blank Filter: If you leave the feature-specification field blank when you press the "Apply" button, then you will be presented with a list of all features. Choose one to use as the filter.

Wildcards: Wildcards can be used on any feature in the filter specification. You can thus type as a shorthand:

- (and mat* abs*)

If any wildcarded feature is ambiguous, you will be prompted with a list of the possible candidates. Wildcards are useful as a shortcut, or in cases where you cannot remember the proper spelling of the feature.

Used in this way, the Review window allows us to locate quickly only those text-items which are of interest. If one needs examples for a linguistic hypothesis, one can quickly obtain all examples from the corpus.

5.1 Functions On Filtered Codings

The Review Interface Menus apply to the current filter-set, rather than to the codings as a whole. Thus, we can select out a sub-corpus, and perform one of the following operations:

- Save selected codings to file: useful for creating a sub-corpus;

- List the text of selected codings: useful when you want to explore a particular category. I may for instance, set the filter to modal-clause, and print the text of these so that I can take them to a word-processor and play around with them.

6 Statistical Analysis

The Coder allows some basic statistics to be performed, mainly descriptive statistics (reporting the means, etc., of each feature), and comparative statistics (splitting the codings into two or more sets, and reporting significant differences between these sets).

For more detailed statistical analysis, the codings can be exported in a form which statistical packages can import.

6.1 Descriptive Statistics

Thee Show Counts option will show the counts and mean value for each feature. The counts apply over the presently filtered corpus, allowing you to get descriptive statistics of subsets of the corpus. These results can be saved to file in tab-delimited form.

6.2 Comparative Statistics

The Comparative Statistics button computes comparative statistics on the presently filtered codings. You will be prompted to choose a system, and the codings are split into a number of sets, one set for each feature in that system. If a coding has feature A, then it is added to set A.

These sets are then compared statistically. The program derives the mean occurrence in each set for each defined feature. These means are displayed in a window, along with the an indicator of

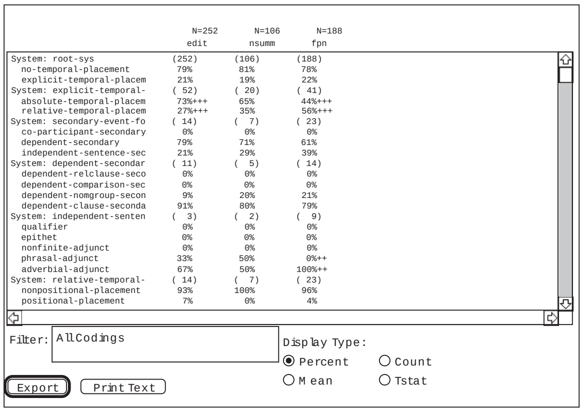

Figure 4: The Comparative Statistics Display

how statistically different that mean is from that in the other sets. See figure ??.

Figure 8:

6.2.1 Significance

The plus symbols after each mean indicate how significantly different the mean is from the means in other sets.

- + Significant at the 90
- ++ Significant at the 95
- +++ Significant at the 98
- (none) Not significantly different.

6.2.2 Local or Global Means

The program adds up the number of occurrences of each feature in the set. The mean value can be calculated in either of two ways:

1. Global Mean: the feature count is divided by the total number of codings in the set. If one root-level feature scored 20which inherits from this feature will add up to 20
2. Local Mean: the feature count is divided by the total number of codings which select from the feature's system. Thus, the means of all features in a system will always be 100

The user can choose between these two modes using the Preferences... option in the Coder menu.

6.2.3 Display options

At the bottom of the Comparative Statistics window, there is a set of radio-buttons, set initially to Percent. Click on one of the other options, and the display will change, to show, rather than percentages, either the mean itself (between 0 and 1), the raw counts, or the t-statistic.

6.2.4 Exporting Results

Pressing the Export button exports the table in a form which can be read into word processing packages. The data is saved tab delimited. I open the file in Microsoft Word, highlight the data (excluding the header data), and select "Text to Table" from the Insert menu. The data is made into a table such as that in table 1.

Tense Editorial Nsumm FPN N 189 80 132 simple-past 21simple-present 42simple-future 11simple-modal 15past-perfect 1present-perfect 11future-perfect - - - modal-perfect - - 1Table 1: Distribution of Tense Selections over Article Types

6.3 Statistical Reports

It often happens that we think we have finished our statistical analysis phase, and we move on to the interpretation of these results. However, often, we may find that our analysis suggests that we need more data of a particular type, that we lack enough instances of one feature to get significant results. So we have to add more data, and do all our analyses again. This also happens as we discover mistakes in our codings, and change some of our feature assignments.

Sick of re-doing all my analyses, I added another functionality to the coder. You can define the set of comparative statistics tests you are interested in (splitting on this system, looking at the differences for these features, etc.). Whenever the data changes, you just load in this file of defined tests, and all the results for the current data-set are printed out to the files you specify. This allows your results to be quickly updated as your codings change.

Reports have the following form:

```
(def-report
  :split-system newspaper-name
  :report-systems (article-type)
  :display-stat :percent
  :local-counting-p t
  ;filter finite-clause
  :save-file "Workbench;TP-Results:1ArtNewFin.lp")
```

Evaluating this will present a comparison over the different newspapers included in the corpus (there were four). It will compare these papers only in respect to their coding of article type (front-page-news, news- summary or editorial). Basically, this report should tell us how balanced our corpus is in respect to having an equal amount of each article type for each paper.

The format of a def-report is as follows:

- :split-system *system* - The system which is used to split the corpus.

- :split-features (*feature1 feature2 feature3* ...) - This form is used instead of :split-system, it is used if you want to compare across only a subset of the features in a system, or if you wanted to compare across features which are not even in the same system.

- :report-systems (*system1 system2* ...) - The systems which are to be included in the report.

- :display-stat *stat-type* - What statistic you want displayed. Use either :percent :mean :count or :tstat.

- :local-counting-p *logical-value* - t if you want local counts (see above), nil otherwise.

- :filter *filter* - a logical complex of features. The codings are filtered on this expression before comparison.

- :save-file *filename* - The file to save the results to. If this field is missing, the results will only be displayed in a window.

6.4 Exporting Codings for External Statistical Analysis

The Coder can export the codings in a form readable by a statistical processor. At present, tab-delimited format is supported. The user can also select which of the features are to be exported, rather than exporting all the data. In our NSF-funded register study, the exported codings are imported into the Microsoft Excel package, or into a statistical package called Statview.

7 Case Study: Choosing Tense in English

To place the use of the Systemic Coder in perspective, I will outline its use in one study, an NSF-funded study into variation of content expression over different text- types (Bateman & Paris 19xx).

7.1 The Phenomena

My role in this study was to examine the variation in the linguistic resources used to temporally place events in different text-types. By temporal-placement, I mean the strategies whereby the writer communicates to the reader the temporal positioning of the event being expressed. Resources for temporal placement include conjunctive relations, e.g., The troops invaded after the bombing. Tense is another major resource for temporal placement, e.g., simple-past tense tends to indicate that the reported event occurred before the time of writing. In this case-study, I focus on the study of tense as a means of temporal-placement.

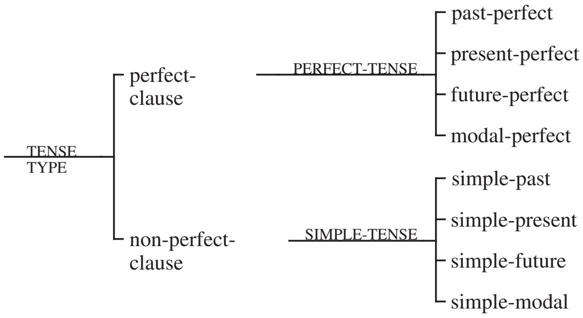

Figure 5: The Tense Systems

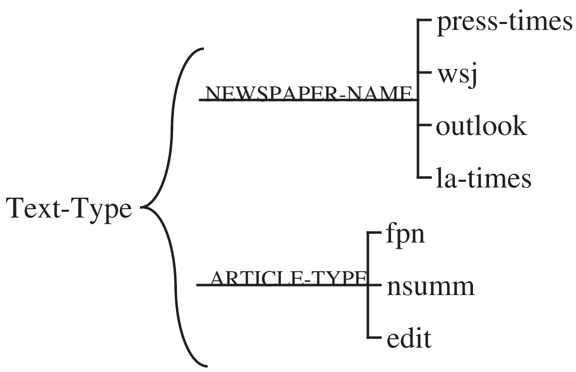

Figure 6: Process Types Informing Tense Selection

7.1.1 Tenses of English

English provides six basic tenses, simple-past, simple-present, simple-future, past- perfect, present-perfect and future-perfect. However, we also need to take into account the possibility of modality – We can bomb Bagdad. Figure ?? shows how the tense systems were organised for this study. The percentage occurrence of each tense in our corpus is also shown. Progressive aspect was ignored.

7.1.2 Text Types

In regards to text-type, three text-types were compared:

- Front-Page-News;
- Editorials;
- News Summary.

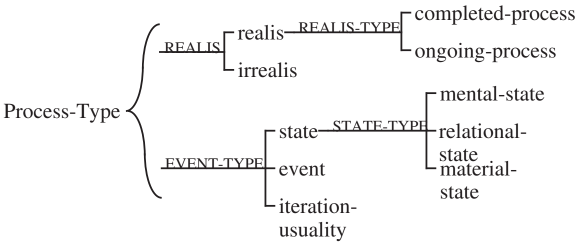

Figure 7: Process Types Informing Tense Selection

7.1.3 Semantic Types Informing Tense Selection

Before beginning discussion of the registerial distribution of the various tense types, we will attempt here to define the semantics of tense – what semantic meaning does each tense choice encode. This discussion will assume two semantic distinctions, shown in figure ??:

- Process Type: Semantic processes can be distinguished between events: "a happening with fixed beginning and end" (Moens & Steedman 1988, p17), states: "an indefinitely extended state of affairs" (ib. id.), and iterations: a repetition of a single process. States may be mental-states (e.g., They hope the war ends soon), relational-states (e.g., Today is the last day of the United nations grace period), and material-states (e.g., Missiles are trained on Bagdad). Iterations may represent a re-occurrence of a process – they bombed 5 times – or a statement of usuality – they usually bomb on Fridays.

- Realis: Processes can also be distinguished on the realis-irrealis axis. Realis concerns whether or not the process has taken place, or is still occurring, or has not in fact occurred. A realis process is one which has either actually occurred, or is still in the process of occurring. The two subtypes of realis are completed-process and ongoing-process. All other processes are labeled irrealis. An irrealis process is one which has not happened, and is not happening now. This includes expectations of the future (e.g., he will run), statements of obligation (he should run), denials (e.g., he did not run), etc. [1]

[1] Note that there is not a clear relation between ongoing-processes and progressive aspect – an ongoing-process is ongoing at the time of speaking, while an event reported in progressive aspect is ongoing at some reference point, which may be the point of speaking, but is often not. For instance, He was running yesterday reports a completed-process (assuming the running finished), but it is reported in progressive aspect.

7.2 Preparing the Corpus

A number of newspaper texts from a range of text-types were entered into the computer. This corpus was then segmented into processes, where a process might be a clause (e.g., They bombed Bagdad), or a nominal-group (e.g., The bombing of Bagdad). After segmentation, the corpus consisted of 700 items.

To support this study, a corpus of 700 processes was coded, exploring various strategies of temporal placement. Of these processes, only 400 are relevant here, the other 300 being either nonfinite clauses or nominalisations, neither of which select for tense. Coding was performed using a program especially developed for coding using Systemic grammar (see O'Donnell 1995).

7.3 Preparing the Coding Scheme

The next step involved the entering of the coding scheme – the system network organising the coding features. Unexpectedly, this phase took up at least as much time as the coding of the corpus itself. Substantial literature surveys were needed into temporal conjunctive relations, temporal aspect and tense. A draft coding scheme was developed. The corpus was scanned visually to see if most data fitted to the draft coding, and when exceptions arose, the coding scheme was modified.

7.4 Statistical Analysis

The next step involves statistical analysis. For this study, we performed a series of comparative analyses, as discussed above, using the Coder's built-in statitistical functions. These results were exported in a tab-delimited format, and then opened within a word-processing package (Microsoft Word). Here, they were automatically reformatted as tables, for inclusions in the NSF report. The results for one particular substudy, for choosing tense in English, are repeated below.

7.5 The Results: Variation of Tense Selection over Article Type

After having presented the various tenses of English, we will now explore their registerial distribution. To simplify discussion, we will focus on one registerial variable – that of article type. Stylistic variation over newspapers will also be examined in a later section.

Since only finite clauses are tensed, the studies below used a sub-corpus, consisting of the 401 finite-clauses in the corpus as a whole.

7.5.1 A Traditional Approach

Traditional studies of English Tense look at the distribution of tense choices over register variation. For instance, Plum & Cowling (1987) studied the correlation of social class, gender, and age with tense selection. They found that, for instance, use of past-tense (primary tense) increases with both age and rising social class. Halliday & James (1993) also looks at the variation of tense selection over differing registers.

We could take this approach with our corpus. For instance, table 3.1 presents the distribution of tense selections in various article-types. This table has one row for each of the eight tenses being considered. Each column shows the percentage of clauses which occur in articles of that type which use the named tense. For instance, 21

The significant result shown in the table is that editorials use far less simple-past than the other article types (21(426

Tense Editorial Nsumm FPN N 189 80 132 simple-past 21simple-present 42simple-future 11simple-modal 15past-perfect 1present-perfect 11future-perfect - - - modal-perfect - - 1Table 3.1: Distribution of Tense Selections over Article Types

These results do not however explain much by themselves. It is up to the analyst to posit some explanation of these results, such that front-page-news tends to express events which have happened already, so simple-past is common, while editorials tend to express the consequences and background of these events, e.g., relational processes such as The United States has friends and interests in the Gulf; The correct policy today is to shift more of the costs of collective security onto our more prosperous allies

However, this is the analyst intruding on the data – the data does not tell us this – all we know directly is the probabilities of particular grammatical choices. The analysis itself has not explained the data, just given the analyst a clearer idea of the patterns which need to be explained.

7.5.2 Separating Content Selection & Expression

One of the main problems with register studies which look at only grammatical choices is that they do not properly separate the differences due to content selection and differences due to content expression. When we start to examine the reasons that the article-types above differ in tense-selection, we start to notice that it is not really a difference in grammatical patterning, but a difference in the types of content that the articles express – the articles differ in typical content, and the difference in tense selection is a result of this, not of a direct register preference for particular tense choices.

For instance, the above data showed that simple-present is the most common tense in Editorials. This does not mean however, that whatever type of event we have to express, we should use simple-present. If we are expressing an event which has already occurred, simple-present is highly unlikely. Simple-past is far more likely.

From this it should be clear that the consideration of tense-selection needs to take into the account not only the register, but also details about the process that is being expressed. For example, whether or not the process has already occurred, what we have earlier referred to as the realis of the process.

What traditional register analyses ignore is that register constrains not only the expression of events within a text, but also the very selection of which events are to be reported in the text – the problem of register needs to be seen under two topics:

1. Content Selection: Which processes and relations are to be reported in the text?

2. Content Expression: How is a given process or relation to be expressed grammatically?

To properly explain the variation in tense across the text-types, we need to separate out these issues. Each will be explored separately below.

7.5.3 Register & Content Selection

Describing register-variation in terms of context-sensitive content selection is quite common in computational linguistics (e.g., Hovy 1988, Paris 1993, etc.). Most of these approaches have assumed a set of knowledge to express (the knowledge base), and attempted to describe how to decide which of the knowledge should be expressed in the text.

We will not, at this point, offer a mechanism for content-selection. Rather, we are trying to describe the register- based preferences for content-selection – what types of content are preferred by each register. It may prove that this information is useful to a content-selection process, but it may not.

Table 3.2 shows how the various text-types vary in their content-selection, as expressed in terms of realis vs. irrealis. These results demonstrate that editorials express far fewer completed-processes (29(40very clear register skewing of content selection.

Feature Editorial News- Summary Front-Page-News Counts: 189 80 32 Realis 60Completed-process 29Ongoing-process 31Irrealis 40Table 3.2: Realis Variation across Article Type

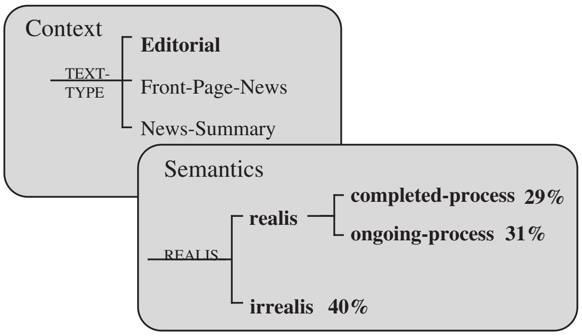

Figure 8: Contextualising Content Selection: Editorials

I have not explored the sub-types of irrealis, since the number of instances in each category is too low to allow reliable results.

Figure ?? shows diagrammatically the results for editorials. This diagram emphasises the relationship between the two axes of table 3.2. We have here a case of register (in this case, article-type) conditioning semantic choice – the frequency of realis types within the text-type.

In summary, the first component of the expanded approach to register analysis looks at register distribution of semantic types. The next section will explore the second component – register variation in the expression of semantic types.

7.5.4 Register & Content Expression

The other half of the problem we need to explore is how the content of the text is realised grammatically, and the role of register in this mapping. For this, we use Semantic Event Analysis, the approach to register analysis introduced by Bateman & Paris (1991). As discussed in chapter 1, this approach explores register variation in terms of the variation in mapping between semantics and lexico-grammar. In other words, given that we have a chunk of meaning to express, how does register influence its expression.

Although this approach to register analysis is relatively unexplored, there is one phenomenon which has been explored in this way: ideational grammatical metaphor. Several studies have explored how semantic processes are expresses grammatically – as clauses, nominal-groups, or adjectivally. Halliday (1985b) for instance, shows that written registers use more grammatical metaphor, and Halliday (1988) shows that scientific discourse also uses more. Eggins et al. (1994) explores the use of ideational grammatical metaphor in History textbooks.

However, none of the previous studies of tense have taken this approach. We will now explore tense selection from such a perspective.

Table 3.3 below shows the mapping between realis and tense, for the corpus as a whole. It shows that realis and tense are strongly interrelated, with four tense strategies realising completed-processes, and two for ongoing-processes.

Tense Complete -Process Ongoing- Process Irrealis N 175 92 134 simple-past 84simple-present - 96simple-future - - 21simple-modal 1past-perfect 2present-perfect 13future-perfect - - - modal-perfect - - 1Table 3.3: Usage of Tense across Realis Types

In the rest of this study, we will ignore irrealis, since there are several sub-types, all of which pattern differently, and we have insufficient data to produce significant results. Also, since there are no occurrences of future-perfect or modal-perfect in the realis data, we will leave these results out.

One fact we can draw from this data is that the realis of a process partially constrains the choice of tense. However, the constraint is not total:

- Completed-Processes: four alternatives are still available:
  - Simple-Past;
  - Present-Perfect;
  - Simple-Modal;
  - Past-Perfect
- Ongoing-Processes: two alternatives are still available:
  - Simple-Present;
  - Present-Perfect;

In the rest of this section, we will be exploring the effect of register on these mappings, in regard to one register variable – article-type.

7.5.5 Article-Type & the Expression of Completed-Processes

Table 3.4 shows the variation in tense selection used to express completed-processes across different article- types.

Tense Editorial News- Summary Front- Page-News N N=54 N=46 N=81 simple-past 67simple-present - - - simple-future - - - simple-modal 4past-perfect 4present-perfect 26Table 3.5: Variation in Expression of Completed-Processes across Article Types

As stated above, four tenses can realise a completed-process. The effects of article-type on this mapping are:

- Simple-Past & Present-Perfect: The major strategies for expressing completed- processes are simple-past and present-perfect, and apart from the simple-modal case, editorials differ in their expression of completed-processes in two ways:

1. Editorials use significantly more present-perfect (26
2. Editorials use significantly less simple-past to express completed-processes (67

As discussed in section 2 above, the present-perfect tense is used when the writer wants to state that the consequences of some past condition are still in force in the present. Since it is part of the role of editorials to relate past events to the reader's present, we would expect a higher incidence of this tense, and this is in fact the case.

- Simple-Modal: The Editorial data contains the only two instances of completed- processes being realised as simple-modals, which was discussed in section 2.2 above. This difference is statistically significant. This is not unexpected, since this is a highly complex tense choice, used to express a past expectation about future events. This type of strategy is more likely in the realm of editorial, which attempt to draw messages from the past, to apply to the present, to influence the future.
- Past-Perfect: Both editorials and front-page-news show a low incidence of past- perfect to express completed-events (3-4this tense option. These results were not statistically significant.

  Figure ?? shows diagrammatically the effect of text-type on the expression of completed-processes. Like figure ??, this diagram attempts to show the contextualisation of grammatical choice – that tense choice is dependent on both current text-type, and also on the content being expressed.

7.5.6 Summary

In this section, we have argued against analysing registerial distribution of grammatical choices in isolation of the semantics they realise. Without taking the semantic context into account, we may reach conclusions about grammatical preferences which are not true grammatical preferences, but rather a reflection of skewed content selection.

To this end, we have explored tense selection in the context of the realis that the tense is realising. We can thus explain the effect of register on tense in two categories:

1. How does the distribution of realis in a text vary with register.

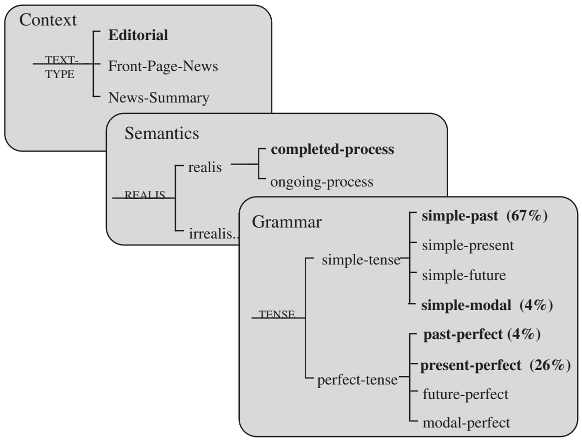

Figure 9: Contextualising Grammatical Selection: Text-Type and Semantics

2. How does the expression of realis in tense vary with register.

The methodology applied here to tense can be extended to explore other areas of grammatical selection, as has already been done in regards to ideational grammatical metaphor.

8 Summary

The WAG Coder is a tool which facilitates the coding of text for use in empirical studies. It semi-automates the acquisition of features. The editing and updating of codings without total re-coding is supported. The coder allows codings to be exported in a form suitable for statistical programs.

9 Bibliography

Alexis, Melina 1996 [paper in this issue].

Bateman, John & Cecile Paris 1991 "Constraining the deployment of lexicogrammatical resources during text generation: towards a computational instantiation of register theory", chapter 5 of Eija Ventola (ed.), Functional and Systemic Linguistics: Approaches and Uses, Mouton de Gruyter, Berlin, New York, pp 81-106.

Bateman, John & Cecile Paris 1989b "User Modelling and Register Analysis: A Convergence of Concerns", Technical Document, Information Sciences Institute.

Bateman, John & Cecile Paris 1989c "Phrasing a Text in Terms the User Can Understand", in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)", Detroit, Michigan.

Bliss, J., M Monk & J. Ogborn 1983 Qualitative Data Analysis for Educational Research, Croom Helm.

Matthiessen C., O'Donnell M, and Zeng L. "Discourse Analysis and the Need for Functionally Complex Grammars in Parsing" in Proceedings of the Second Japan-Australia Joint Symposium on Natural Language Processing, October 2-5, 1991, Kyushu Institute of Technology, Iizuka City, Japan.

O'Donnell, Michael 1994 Sentence Analysis and Generation - A Systemic Perspective. Ph.D., Department of Linguistics, University of Sydney.

O'Donnell, Michael 1995a "From Corpus to Codings: Semi-Automating the Acquisition of Linguistic Features", in Proceedings of the AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation, Stanford University, California, March 27 - 29.

O'Donnell, Michael 1995b "Sentence Generation Using the Systemic WorkBench", in Proceedings of the Fifth European Workshop on Natural Language Generation, 20-22 May, Leiden, The Netherlands, pp 235-238.

Webster

Paris, Cecile & John Bateman 1990 "User modeling and register theory: a congruence of concerns", Technical Report, USC/Information Sciences Institute.