Chapter 3: Coding and extracting data

Monika Schmid

Marjolijn Verspoor

Brian MacWhinney

Introduction

When investigating SLD, a usage-based and dynamic systems point of view proceeds from assumptions which are different from those of traditional approaches. DST studies are not so much interested in monitoring whether the L2 learner converges towards the native norm in the data he or she produces but in how the language system develops over time. In particular, the notion that a complex dynamic system consists of subsystems which are never entirely stable and may exhibit a great deal of variability, particularly during stages where the whole system is undergoing intensive development, is relevant here. This entails that DST approaches to SLD cannot confine themselves to data gathered on the development of one linguistic structure or rule but must take a broader perspective on the full range of the linguistic repertoire of a learner.

Such a perspective cannot be gained on the basis of many of the traditional experimental methods of research in Applied Linguistics. While specific tasks that are often used to gauge the state of an individual learner or a group of learners may provide additional insight into what is going on with respect to the acquisition of some specific feature(s), the full range of the linguistic repertoire can only truly be investigated on the basis of (spoken or written) data produced under relatively natural conditions - that is, data where all aspects of the linguistic

production process (the selection of the vocabulary, the sentence frame, grammatical aspects such as tense, mood and voice, orthography or phonology and so on) are, as far as possible, fully under the control of the learner.

The reason for this approach is that, while the process of SLD usually implies an overall increase in linguistic knowledge, accuracy across all linguistic levels and complexity of vocabulary and style, there may be trade-offs between the individual components of language, in particular in situations where there is intensive development of one of these components. For example, a student who has previously shown an exceptionally broad vocabulary may, at some point in time, appear to regress with respect to lexical sophistication. This 'dip' in her development may appear puzzling if the other aspects of her linguistic repertoire are not taken into account. However, when a full analysis is carried out, it may appear that at the same time, there is a marked increase in the length and complexity of the sentences she produces. In other words, at this point in time the student is diverting a large part of the limited cognitive resources she has available for the production and finalizing of messages in her second language to the syntactic component of her linguistic knowledge, and therefore does not use the full range of vocabulary skills which she obviously has had at her disposal, as an analysis of her previous (and subsequent) performance can show. This tradeoff would not have been apparent in a test tapping only into her vocabulary knowledge. Below, we will present an analysis of an individual case of SLD based on a sequence of written texts produced over a longer learning span, and show points at which such tradeoffs emerge from the data.

Before that, however, we will address the issue of how to analyze free data. It has to be acknowledged that this is a highly costly and effortful process. The reason why researchers often prefer to rely on experimental data elicited by means of a specific test is that these can be gathered from a group of second language learners at the same time (exam-style), be coded

quickly and dichotomously (in terms of correct or incorrect responses), and be quantified in terms of percentages of accurate responses which are easy to analyse statistically.

Free data, on the other hand, present a number of obstacles. Firstly, in particular where the researcher wants to rely on spoken data, they have to be elicited from each learner separately, either by means of a free interview or (if it is the aim to keep the topic constant across the data) by a task involving the description or re-telling of a stimulus such as a picture or a film. This already means that the process of collecting the data may be far more time-intensive than where controlled experimental tasks are involved. Secondly, the transcription of such data is also invariably an extremely time-consuming process.

Both these obstacles can be circumvented to some degree by setting learners a specific topic and asking them to write a certain amount of text on this matter. This, of course, is a solution which may not be applicable in all cases, as written and spoken data are by no means equivalent expressions of a learner's linguistic knowledge and development. In either case, however, any kind of investigation which attempts to deal with naturalistic data, be they spoken or written, will have to expend a great deal of thought, time and care on the process of coding. This is particularly true for investigations such as the ones that are carried out from a DST perspective, which attempt to paint a full picture of the development of an individual's proficiency across all components of their linguistic repertoire.

In order to capture this process, the coding of linguistic data cannot be confined to the simple counting of errors. Nor may it even be enough to calculate accuracy as a function of the proportion of obligatory contexts, since an increase in proficiency - particularly at higher levels - often involves options which are stylistically determined, for example the use of the passive voice. This means that a large number of features may have to be coded not only across all obligatory, but across all possible contexts.

Coding and extracting data

Fortunately, we no longer live in an age of typewriters or abacuses. A good understanding of the available computer resources can help us make onerous processes of transcription, coding and analysis far more efficient. In the following, we will take you through a case study. First we will look at the raw textual material, discuss what measures we decided to look at and then explain step-by step how we coded, extracted and visualized the data.

The case study: measures to be analyzed

The corpus investigated here consists of 22 text files collected over a four-year period from an advanced student of English who majored in English at a university in the Netherlands. Her first language, Dutch, is typologically similar to English, and after six years of English instruction at high school she was quite an advanced user of English even when she entered university. She attended classes in academic writing, literature and linguistics and for each of these classes had to hand in written assignments. The texts were on different topics, such as literature, culture and society, or linguistics, but they were all written in under similar circumstances (at home without time pressure) in an academic register. We took a random sample of about 200 words from each text (containing only full sentences) and ordered these excerpts chronologically. These data were previously investigated by Verspoor , Lowie and van Dijk (2008). The analyses presented here, however, differ from the original ones in a number of ways - chiefly in that they rely more on programs and automatized functions. Below are parts of three samples, the first assignment, one from the mid-range (13th), and the final one (22nd).

1. Practising an exam question for 20th century English fiction

The story is about the relationship between a father and his two daughters and about the way he influenced them. After their father's death, the two grown-up sisters, Constantia and Josephine weren't controlled by their father anymore and also they couldn't rely on him anymore. They were forced to make more decisions on their own now, in which they didn't succeed. They couldn't even decide on trivial matters like whether they would like their fish to be boiled or to be fried. Even at the end of the story, they couldn't make up their minds about ordinary matters. The story ends with a dialogue between the two women about the question whether they should keep Kate as a maid or fire her. The father had always prevented them to make decisions on their own.

13. The master and his slave

As the relationship between Prospero and Ariel is not equal, it is not perfect either. In the first part of the scene, it becomes clear that Ariel wants to fool Prospero by letting him know in an ironic way what he thinks of Prospero's demand. The absurdity of all the things he says to be willing to do for his master in a dramatic way give his statement ironic overtones. The strongest evidence that Ariel and all his quality are not to Prospero's bidding task comes from the end of the play where Ariel asks in a way of indignation. The term of toil already tells us that the does not like working for Prospero at all, since it stands for hard, unpleasant work. Without knowing what it is that Prospero wants he does not like the idea of helping him again, because he does not like to endure another hardship. In other words, he does not really want to dive into the fire for Prospero.

22. Task-based language learning and teaching

> Focused tasks are tasks that elicit use of specific linguistic features, either by design or by the use of methodological procedures that focus attention on form in the implementation of a task. One of the major purposes of these tasks for learners is to induce their incidental attention to some specific linguistic forms when processing either input or output. Focused communicative tasks involving both reception and production (the main focus of this chapter) also serve clear goals for both researchers and teachers. For researchers, they provide a means of measuring whether learners have acquired a specific feature. They are often preferred to tests because they provide evidence of what learners do when they are not consciously focused on using a form correctly and thus can be considered to elicit implicit knowledge rather than explicit knowledge. Many SLA researchers would consider that only when learners demonstrate they are able to use a feature spontaneously in communicative activity can they be said to have acquired it. Focused tasks are of value to teachers because they provide a means of teaching specific linguistic features communicatively – under 'real operating conditions'.

Even a cursory glance at the samples reveals that at the end the writer uses more complex language. At the beginning the sentences are shorter (from an average sentence of about 18 words at first to 23 and 26, respectively). The early sample also contains many dependent clauses with finite verbs, often with a pronoun (*he* and *they*) whereas at the end other types of complex constructions appear: the noun phrases are longer (e.g. the noun phrase *tasks that elicit use of specific linguistic features, either by design or by the use of methodological procedures that focus attention on form in the implementation of a task* contains 29 words). Furthermore, there are many non-finite constructions like ***to induce*** *their incidental attention to some specific linguistic forms when* ***processing*** *either input or output.* The vocabulary also appears to have become more advanced. For example, in the first text the writer uses rather

every-day, familiar words like *influence, force* and *control*, but in the last text there are less frequent words which are more typical of the academic register, such as *elicit, induce,* and *acquire.* To what degree the familiarity and frequency of lexical items differs can, for example, be established using the British National Corpus (BNC) online (http://www.natcorp.ox.ac.uk/): here, the three verbs cited from the first text all score between 10,000 and 30,000 hits, while the three verbs from the last one range between 250 and 2000, and so are used far less frequently overall. We will discuss ways of measuring such factors below.

Our main research question from a DST perspective is how this development took place. Do all the variables that indicate increased complexity develop simultaneously, or do they compete? For example, how and when do the finite dependent clauses decrease and how and when does she start using longer noun phrases and more non-finite structures? Do these variables develop slowly and smoothly or do they show peaks and troughs? Another question is to what degree a more sophisticated lexicon and sentence complexity are related. Does the appearance of a more academic vocabulary consisting of overall less frequent items precede more complex syntactic structures, or do they go hand in hand?

To answer these questions we have to examine the relevant variables. In order to do this, we first have to decide which variables may be relevant and how we can 'operationalize' them, ie. make it possible to count and quantitatively evaluate them. However, we may not know until we have looked at the different variables which ones are most relevant. Our coding procedure should therefore cast the net as widely as possible, to allow us to determine which variables are useful for our analysis. (Note that it is easier to code too many than too few things.)

We know from previous studies (e.g. Wolfe-Quintero, Inagaki, & Kim 1998) that sentence length is a usually a good indicator of complexity (the longer the sentence, the more

complex it will typically be), the average number of words per sentence/utterance may therefore be a useful variable. However, sentence length alone tells us nothing about the way the sentence has become more complex. We know that beginners use more simple sentences and advanced students more complex ones. In addition to sentence length, we therefore should code sentence type: simple, compound, complex and compound-complex. Previous studies (cf. Wolfe-Quintero et al.) have also indicated that more advanced language learners use more dependent clauses, so these should be coded as well. However, as we are specifically interested in the developmental process (which types of clauses occur first, which later), we will code different types of such finite clauses: adverbial, relative and nominal clauses (the latter are clauses used as subject or object or after a preposition). The text samples analyzed here also showed an increase of non-finite structures towards the end of the developmental period we observed, so these are coded separately.

In addition to these measures of sentence complexity, we also want to establish development in the lexicon/vocabulary. There are various approaches to this. A very simple way of measuring lexical diversity is to calculate the so-called type-token ratio (TTR). This is simply the number of *different* words (types) contained in a certain stretch of text, divided by the *total* number of words (tokens). For a stretch of text in which each word occurs only once, the TTR would therefore be 1. However, straightforward type-token ratios can be problematic due to the presence of *function words* (e.g. determiners, prepositions, adverbs) which are highly frequent. This means that, as texts become longer, the type-token ratio invariably goes down. A number of different ways to compensate for this have been devised and are described in Vermeer (2000). We shall confine the discussion here to one such measure, the so-called D, which is based on the probability of new words appearing in longer and longer stretches of text. The precise calculation of D is rather complicated (for a full discussion see McKee, Malvern & Richards 2000) - fortunately, there are programs which can perform the analysis

for you automatically. One such tool, the CLAN program, which was developed in the CHILDES project (http://childes.psy.cmu.edu/), will be discussed in detail below.

In addition to purely quantitative measures that consider some kind of relationship between total words and total different words, there are a number of other indications of lexical diversity. Consider the following two stretches of text (from Jarvis 2006):

> Text 1:
>
> There was a girl who was alone and hungry. She stole some bread from a bakery and tried to run away, but she ran into a man, and they both fell down. That gave the police enough time to find her and catch her.

> Text 2:
>
> A destitute and lonely young female stole a loaf of bread from a bakery. She attempted to flee, but she collided with a man who was walking toward her, and both of them fell down. In the meantime, a policeman arrived and detained them.

As Jarvis points out, both utterances contain exactly the same number of tokens (44) as well as types (35), so that the TTR of both texts is identical. However, it is obvious that the second text uses lexical items which are less frequent to describe the same set of circumstances (e.g. “tried to run away” vs. “attempted to flee”) in a manner that is similar to the lexical development pointed out for our learner above. In line with suggestions by other researchers, e.g. Laufer & Nation 1995), Jarvis therefore suggests that, in addition to ‘blind’ measures of lexical diversity, such as TTRs and D, other factors, such as the rarity of the items used, should be considered. From previous studies (cf Grant & Ginther 2000) we know that, as proficiency increases, L2 writers compose longer essays with more unique words, and, on the

average, longer words. More proficient writers also use different types of words (relatively more adjectives, adverbs and prepositions). We will look at all of these to help us chart the path of development of our advanced writer.

Another measure that may tell us something about more sophisticated vocabulary is word length, as more proficient writers tend to use longer words at least in English (Grant & Ginther 2000). One of the reasons for this is that less frequent, more academic words tend to be longer. For example, in Text 1, the average word length is 3.7 characters and in Text 2 4.4 characters. Such an analysis, however, also has to take into account issues of word class. In Text 1 there are more function words such as pronouns, prepositions, helping verbs like *to be*, and so on, which tend to be shorter than content words such as nouns, verbs, adjectives and adverbs. Text 1 has 16 content words and 28 function words, whereas Text 2 has 20 content words and 24 function words. If average word length is calculated on the basis of content words only, the difference between the two texts is even clearer: Text 1 has an average word length of 4.62 and text of 6.35 characters. It is therefore important to distinguish between content and function words in analyses of word frequency and word length, since a higher rate of function words (which might also be associated with syntactically more complex style) can quickly confound the issue.

Table 1 gives an overview of all the variables that we will be looking at in the texts of this advanced writer. Note that the definitions must be very specific in order to identify these variables consistently.

General sentence complexity	Average number of words per utterance	Number of words in the text divided by the number of utterances.
	Finite-verb ratio	Number of words in the text divided by the number of finite verbs in the text.

Sentence complexity	Simple	Sentence contains one finite verb (may include non-finite structures)
	Compound	Sentence contains two main clauses, each with its own subject and finite verb (other compound structures are ignored)
	Complex	Sentence contains at least one finite dependent clause.
	Compound complex	Sentence contains at least one dependent clause and compound structure at the sentence level or clause level.
Dependent clauses	Finite Nominal	Clause functioning as subject or object (also a clause after a preposition)
	Finite Relative	Clause functioning as post modifier of a noun (also an appositive clause)
	Finite Adverbial	Clause functioning as adverbial
	Non-finite (all)	Non-finite constructions functioning as nominal, adverbial or modifier
Lexical measures	TTR	The number of Types divided by the number of Tokens
	D	A randomized Type Token ratio
	Unique lexical items	The proportion of unique words per text
	Average length of lexical items	The average length of content words (all function words are ignored)
	Frequent lexical items	The proportion of very frequent items

Table 1. Overview of variables to be traced in the case study of the advanced writer.

The above overview demonstrates that the number of variables to be considered in our analysis is quite large. Given that our corpus is limited to 22 texts of about 200 words each, it might still be feasible to code all of these by hand. However, linguistic corpora usually contain more and longer texts, where such a coding procedure would be extremely time-consuming at best, and impossible at worst. The purpose of the following sections is therefore to demonstrate to what degree the process of coding and analysis can be automatized in order to make it more efficient.

For this process, we will avail ourselves exclusively of tools and programs which are either widely available (such as Microsoft Office) or can be freely accessed and downloaded from the internet. In particular, we will introduce a set of conventions for the formatting and coding of data, known as the CHAT format. This format has been developed in the CHILDES (Child Language Data Exchange System) project described in MacWhinney (2000), together with a set of tools for analysis which can be applied to any text which is set out according to the CHAT conventions, some of the functions offered by the Microsoft Office package, and the use of spreadsheet software (Microsoft Excel) for the further analysis of the output produced by the CLAN program.

In the following sections, we will first describe how to set up texts in CHAT format and how to set up CLAN and check your files through the CHECK command. Then we will explain how you can create your own set of tools to code data and use CLAN to perform frequency and morphological analyses. Finally we will discuss how all the data may be visualized in charts.

Formatting in CHAT

The method of coding and analysis described here was originally developed for the study of child language, but has since been widely used, adapted and applied for bilingual development as well: the Child Language Data Exchange System (CHILDES), its set of conventions for the transcription of data (CHAT) and the tools provided for the analysis of data which have been transcribed according to those conventions (CLAN).

The CHILDES project provides a standardized system for the transcription and analysis of data which emerged from the desire among researchers on child language to be able to share and exchange their data. The CHILDES internet resource (http://childes.psy.cmu.edu) contains, among other things, a substantial database of transcripts as well as audio- and video files from a large number of language acquisition contexts, the manuals for both CHAT and CLAN, the CLAN program and a set of tools for morphological analysis. All of these resources are free.

Since extensive and detailed documentation is available at this location, we will keep the description of the CHAT transcription system and the use of the CLAN program to a minimum here. Apart from acquainting you with the basic conventions and commands necessary to work with these resources, we will focus largely on ways of developing your own tools for coding your data within the CHAT files.

CHAT files are text-only files which contain linguistic data in a format that the CLAN program can read and analyze. For researchers working with audio files, CLAN may be used in what is referred to as 'coder mode' from the start. This will ensure that all transcriptions conform to the CLAN standards, and eliminate many potential problems. However, if you are working with files which were written using a different text editor, such as Microsoft Word or WordPerfect, you will have to convert those files to CHAT format and text-only mode. This conversion can result in a number of problems, since many text editors introduce automatic formatting to texts, for example replacing straight quotation marks or apostrophes (') by what

they sometimes refer to as 'smart' quoation marks (‘ or ’). When the file is converted to text-only, these and other symbols (especially the diacritics used in many languages) can be changed so that they are not recognizable to CLAN or other programs. These problems can be further aggravated if the software used by the original writer of the text is different from the one which you are using (e.g. if one is an Apple user and the other a Windows person).

The CHAT files must be formatted carefully in order to conform to certain standards which the CLAN program recognizes. Here is an example of what a CHAT-file might look like (the symbol → represents a tabulator):

```
@Begin
@Languages: en, nl
@Participants:  XYZ 301138 Student
@ID:  en, nl|corpus|XYZ|||3011||Student||
*XYZ: i like my new school and i love my new friends.
*XYZ: i do al [*] lot with Shanice.
%err: spelling
*XYZ: she sit [*] next to me in the class.
%err: verbagreement
*XYZ:  <with Inge, Iris have i got> [*] a lot of fun to
[*].
%err: wordorder, spelling
*XYZ: only the bags are very heavy to wear [*].
%err: semantics
@End
```

At first sight, this text may appear rather complex, but once you have familiarized yourself with the CHAT coding system, you will find that it quickly becomes easy to use, as long as you observe a number of principles strictly. A CHAT-file is a text-only file with the extension .txt, .cha (this denotes a CHAT transcript) or .cex (identifying a file on which some kind of CLAN operation has been performed). If you prefer to work with a text editor such as Microsoft Word, it is important that you save the files as plain text. Each CHAT file contains data from one data collection session. Usually, this is data which was originally in the spoken form and has been transcribed, but CHAT/CLAN has also been used for the coding and analysis of written data.

CHAT files are organized in different 'tiers' or layers, the three most important of which are header tiers (in the example @Languages, @Participants, @ID), utterance tiers (in the example *XYZ) and dependent tiers (in the example %err). Note that the codes for these tiers (e.g @Languages) must be followed by a colon (:) and a tabulator (→).

In the following, you will find more detailed descriptions of the conventions, rules and restrictions applying to each type of tier.

Header tiers

Header provide general information about the file (the speaker(s), the language(s) used etc.) and must start with the symbol @. This symbol helps CLAN realize that the line does not contain actual linguistic data, but meta-information.

- @Begin and @End

  CHAT files begin with a line which contains the sequence "@Begin" and end with a line which contains the sequence "@End". No empty lines are allowed either before @Begin after @End (nor anywhere else in the file).

- @Languages:→

  Here, you code which is the main language of the text. If there is more than one language, you can specify this as well, as was done in the above example. In this particular case, the text is in English, which is therefore identified as the primary language. Dutch is coded as the second language, because a number of texts in this corpus contain some code-switches. In the actual text, words which do not belong to the main language can be identified by means of the suffix "@s". A list of abbreviations for different languages can be found in the CHAT manual.

- @Participants

  In this line, a three-letter code must be specified for each person who participates in the interaction transcribed in the file. The code is used to identify the speakers for each utterance in the main body of the text. You may choose a code which identifies each individual speaker, e.g. the first three letters of his/her name, but we have found it useful to pick the same code for the same type of speaker (the student, the investigator etc.) across all the files in a particular corpus. This is helpful later on, when you want to have CLAN analyze the utterances of the people you are interested in (and excluding others, for example, the interviewer) for a whole set of files in one go.

  One option is to choose codes symbolizing the role of each speaker, e.g. STU for student, TEA for teacher, INV for investigator, CHI for child, and so on. We prefer a combination of letters that will appear nowhere else in the text, that is, something that will definitely not be part of any word, for example XYZ. This can sometimes make automatic search-and-replace routines easier.

  This three-letter code is (optionally) followed by the name or identification code of the person. In our case, it is the identification number which we have given each student. It could also be a name (note that no spaces or punctuation are allowed, so the name should

be e.g. Monika_S_Schmid).

The name is followed by the 'role' of this speaker, in the present case 'Student'. The CHAT manual lists a large number of possible roles (Child, Mother, Adult, …). Note that roles are case-sensitive, so the role 'student' with a lowercase s will lead to an error message in CLAN.

@ID

The identification tier specifies slots for the following information:

@ID:→**language(s)**|**corpus**|**code**|age|sex|**group**|SES|role|education|.

The information set in bold above is obligatory, all other information is optional. In our example, the non-obligatory information has been omitted:

@ID:→en|corpus|XYZ|||3011||Student||

You may chose not to code the ID tier initially. Once you perform the command CHECK in CLAN (see below), the ID tier will be inserted, based on the information in the @Participants line, if it is found to be missing.

In addition to these obligatory headers, there are optional headers, e.g. @Coder (to enter the name of the person who did the coding, useful if several people work on the same project), @Activities (to describe the situation which was recorded), @Comment, @Date, and @Location. For a full list, and how to fill these, see the CHAT manual.

Utterance tiers

Utterance tiers identify the person who has produced this particular utterance and contain the actual linguistic (spoken or written) data. They must start with an asterisk * followed by the three-letter code identifying the speaker which was specified in the @Participants tier. Note that the three letter code must be followed by a colon and a tabulator (strictly in that order, and with no empty spaces intervening anywhere). Every utterance consists of one sentence:

```
*XYZ: → hello mij holiday was great.
*XYZ: → i have been in Spain a beautifull country!
*XYZ: → i'ts hot in there, thirty degrees.
*XYZ: → i was every day on the beach, and now i'm brown.
```

The lexical analyses which can be performed by the CLAN program can, of course, only recognize standard spelling. Deviations, like *mij* (instead of *my*) or *beautifull* will not be identified correctly. We may want to code these errors in the dependent tiers in order to perform an extended error analysis, but for CLAN to be able to ignore these misspellings (or repetitions, retractions, pauses etc. in spoken data) and correctly count the lexical items used, we must provide appropriate information within the utterance tiers themselves.

Errors in the utterance are identified by an asterisk enclosed between square brackets [*]. If the error extends over more than one word (as in the phrase 'with Inge, Iris have I got' above), the entire error is furthermore enclosed between angled brackets. Note that it is important to ensure that each opening bracket has to have a closing bracket, […] or <…. > on the same utterance tier. An error can therefore not extend across several utterances.

```
*XYZ: → hello mij [*] holiday was great.
*XYZ: → i have been in Spain a beautifull [*] country!
*XYZ: →  i'ts [*] hot <in there> [*], thirty degrees.
*XYZ: → <i was every day on the beach> [*], and now i'm
brown .
```

In order to allow for more detailed coding of the errors, the utterance can be followed by a dependent 'error tier' (%err: →) to be discussed below. If, in addition to marking these mis-spellings as errors, you want to enable to CLAN program to include the words produced here in frequency analyses, you can provide the correct alternative for a misspelling together with the error code:

```
*XYZ: → hello <mij> [* my] holiday was great.
*XYZ: → i have been in Spain a <beautifull> [* beautiful]
country!
*XYZ: →  <i'ts> [* it's] hot <in there> [*], thirty
degrees.
*XYZ: → <i was every day on the beach> [*], and now i'm
brown .
```

- code-switches are indicated by the symbol @s following the word:

  ```
  *XYZ:  I don't like cheese but I do like honing@s and
  bitterballen@s very much!
  ```

  If an entire utterance is switched to the secondary language of the text, this can be indicated in the following way:

  ```
  *XYZ:  [- en] ik vind het heel leuk op deze school.
  ```

  (I find it very nice at this school. - "I like this school")

  In other words, a code is included at the beginning of the utterance informing the program that the present utterance is not in the primary language of the text (the one which was specified first in the @Language tier). Should an utterance be predominantly in Dutch, and be identified as such by means of the [- en] code, but contain a few English words, then these can in their turn be marked with the @s suffix.

Note that sometimes material can be ambiguous. In the present example, the researcher has to decide whether the word *mij* (pronounced /ma□/ in this student's first language) is an orthographical error (a misspelling of *my*, since in Dutch the letter combination *ij* is interchangeable with *y*) or a code-switch[1]:

```
*XYZ: → hello <mij> [*] holiday was great .
```

or

```
*XYZ: → hello mij@s holiday was great .
```

As the present chapter is dealing with the analysis of written data, these are the only codes pertaining to the utterance tiers which we need to concern ourselves with. The CHAT manual gives in-depth descriptions of a number of other codes, pertaining for example to the treatment of repetitions, retractions, interruptions and self-interruptions, overlaps and so on. There are a few other important considerations with respect to the utterance tier that must be observed before CLAN can perform data analysis operations on the files.

Utterance delimiters: Each line must end with an 'Utterance delimiter', which can be a period (.), an exclamation mark (!) or a question mark (?). (This also implies that the utterance delimiter may not appear within an error code: verry. [*] should be verry [*] .) Note that utterance delimiters may appear only at the end of the utterance. In other words, no utterance delimiters are allowed within a line and only one sentence may be used per utterance line.

Punctuation marks: All other punctuation marks such as commas (,), semicolons (;) or brackets at the end of a code (], >) must be followed by a space. To make sure you have

[1] In our study we considered it a misspelling if the word occurred in an English context and a code switch in a Dutch context.

spaces after each punctuation mark, you might do the following. When you are finished coding your file, run a search-replace (Edit – Replace). In the "Find what" line, enter just a comma, in the "Replace with" line enter a comma followed by a space. Then do the same with the brackets] and >. This, of course, means that those symbols which were already followed by a space prior to your search-replace are now followed by two. This does not really matter, but if you want to be very tidy, you can afterwards run another Search-Replace, where you replace two empty spaces by one.

Abbreviations: Abbreviations may not be followed by an utterance delimiter. Since you may not use periods anywhere except at the end of the utterance, you cannot use abbreviations such as Mr. or Mrs. and these words have to be spelled out (the CHAT manual suggests *Mister* and *Missus*). Individual letters and abbreviations, such as P.E., have to be coded by following each letter with a @ and lowercase letter L, i.e. "P@l E@l" (Note that there is no space between the letter and the @ or between the @ and the l, but there has to be a space between the code for each individual letter (i.e. between P@l and E@l etc.). Acronyms, such as USA are represented as U_S_A.

Capitals and numbers: The use of capitals and numbers is restricted. For example, uppercase letters within words are not allowed. This can sometimes be an error source when there is a space missing between words, e.g. writeEnglish. Names such as McDonald would also be rejected. Numbers have to be spelled out (not 4 but *four* - note that in our data, we simply replaced all numbers with the word "numb" in order to ensure that these items would not distort lexical frequency counts). Furthermore, the program for morphological analysis available in the CLAN program automatically identifies all words which start with a capital letter as proper nouns; you should therefore use lowercase initial letters for words which are

not nouns or names, even at the beginning of sentences or for the first person singular pronoun. Titles or other stretches of text which use non-default capitalization can be excempted from this convention by replacing the space between the words with an underscore, e.g. The_Tempest.

Dependent tiers

Each 'utterance' line can have several dependent tiers pertaining to it, where additional information which you wish to analyze may be coded. These are preceded by a % symbol, plus a three-letter code indicating which information they contain, such as the %err tier included in some of the examples above, where information about the errors contained in the previous utterances is specified.

You might also have a %com tier, where you can enter comments (such as 'the telephone rings'), a %spa tier where you code speech act information and so on. You may not have more than one tier of the same type per utterance, so if there is more than one error in the same utterance, they all have to be coded in the same %err-line. The CHAT manual contains a list of pre-defined dependent tiers. Should these not be sufficient for your own coding needs, you can add your own three-letter tier names and prefix them with the letter x, as we have done here.

The option of creating dependent tiers to accompany each utterance is what makes the CHAT transcription system such a powerful and useful tool for the analysis of learner data, in particular for investigations from a DST perspective. It allows the researcher to tag utterances for any number of characteristics, and then relate these to each other, without making an unreadable mess of the utterance itself. With the help of the CLAN program, frequency analyses can be extended or limited to any type of tier - the utterance tier, a particular set of dependent tiers, but also dependent tiers of a specific kind, such as a specific error code.

For example, if we do assume that in the process of development, there may be a tradeoff between subcomponents of linguistic skills, such as pronunciation, syntactic complexity and lexical richness, we might create a pronunciation tier where the target-likeness of the production of each instance of a particular phoneme is coded. Each utterance might also have a syntactic tier associated with it, where we specify sentence complexity. We can then analyze these two features in relation to each other, and also identify lexical richness measures, such as type-token frequencies, in utterances of varying syntactic complexity or phonological accuracy. That is, we may trace the development of a speaker not only across data collection points, but within a particular session.

Setting up CLAN and checking your files through the CHECK command

Once you have prepared all of your transcriptions in text-only files according to these rules, you are ready to begin using the CLAN program. This program can be downloaded freely from the CHILDES website and installed on your computer. Before you can analyze your data, you have to make sure there are no errors in your files.

Setting up CLAN

When you start the program, you will see a screen looking more or less like the one depicted in Figure 1.

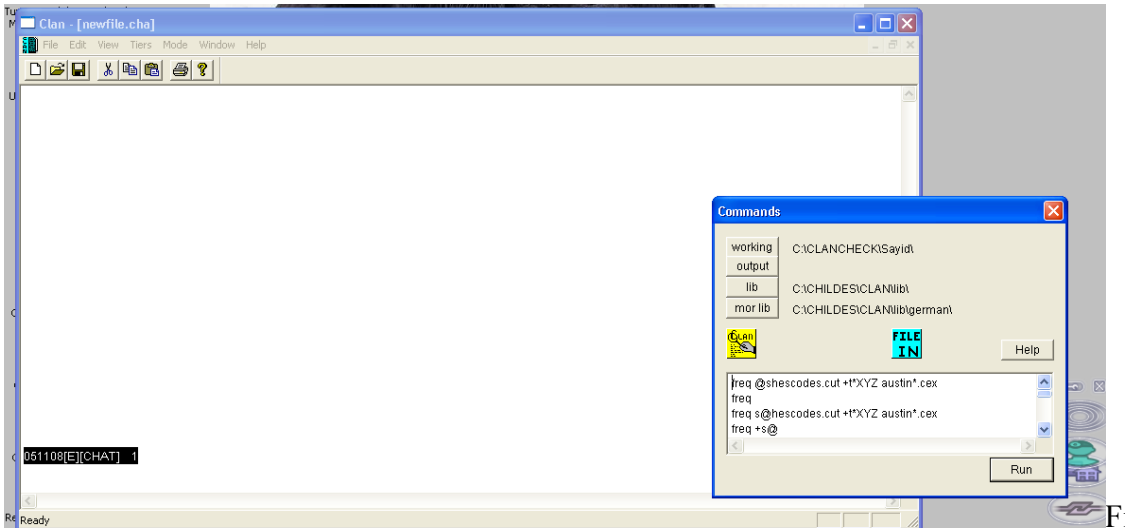

Fig. Figure 1: The CLAN program

The large window is where the output from the commands will appear; in the small window, you can enter the commands. But first, you have to tell CLAN where to look for your files. Click the button ‘working’. This will bring up a dialog box from which you can select both the drive and the directory where the files which contain your data are located. If you want a different directory for the output that will be created from the analyses you plan to run, you can specify this under the 'output' button.

The CD included with this book contains a corpus of 22 text files which have been coded in CHAT. There are several folders containing these files at different stages of the coding process. For the purpose of the exercises below, you can insert the CD and set your working directory to the folder "initial". You also have to specify a folder on your computer as Output folder, since the CD is read-only, and no output files can be written to it.

The two buttons left are 'lib' and 'mor lib'. Here the folders for program help files are specified - files which are used as the basis for the commands that will be run. By default, 'lib' is in the folder in which the CLAN program was pre-installed (e.g. C:\CHILDES\CLAN\lib), and it contains a number of files necessary to run the program, which usually have the file extension '.cut'. The most important of these files is the 'depfile.cut'. In this folder you can

also store files which contain lists of items that you wish to be counted or changed in all of your data files (for details see the CLAN manual).

The directory specified under 'mor lib' is where the files which are necessary to perform a morphological analysis are stored. CLAN offers the possibility for morphological tagging of each word in a data file for a number of languages - among others English, Spanish, German, Dutch, Hebrew and Cantonese (see http://childes.psy.cmu.edu/morgrams/). For these languages, other researchers have written grammars and compiled lexicon files which can perform a morphological analysis of each word, identifying the lexicon entry form (the 'lemma') from which it was derived, the word class, and the inflectional form. If you want to avail yourself of this function, you may download those files separately and specify under 'mor lib' where the directory is which contains them.

Checking your files

Before you can begin working with CLAN, you have to verify whether there are no problems or oversights left in your coded files. If your working directory contains a set of files ending with the extension '.cha', as is the case in our example, the command for checking these files is "check *.cha" (in other words: check whether all the files located in the working directory which have the extension .cha are in compliance with all necessary CHAT transcription conventions).

Note that if you are using a text editor such as Word for the transcription and coding of your files, you have to make sure that all files are closed in Word before you check them in CLAN. Word creates backup files for any open files which contain a large amount of information pertaining to the file. This information is invisible in Word, but it will upset CLAN. If an open file in Word is the source of an error, you can identify this by the file name to which the message pertains. Here is an example of such an error message:

```
From file <E:\MY FILES\Texts\~$01.cha>
*** File "E:\MY FILES\Texts\~$01.cha": line 1.
```

You can see that the filename which appears here starts with the symbols "~$". This means that this file is the backup of a file opened in Microsoft Word. These backups are erased when the file is closed (note that in the case of a program crash, the backup files remain and later have to be deleted by hand).

If you are working with files that were created with a different text editor and then converted to text-only format, the initial results of the CHECK command can sometimes be very discouraging with huge numbers of mistakes, but deciphering and correcting them is not too difficult once you know what to look for. The folder "checkfile" on your CD should contain two files which create problems, and one which checks out okay. You can verify this by setting your working directory to this folder, and then typing 'check *.cha' at the command prompt.

There are two types of errors at this stage: "Basic Syntax errors", where the violation detected in the file refers to the basic coding conventions, so that no further checking of the actual text is performed, and ordinary errors. A Basic Syntax error is caused, for example, by the use of tabulators within utterances, or by the non-use of tabulators after the utterance code. To illustrate this, we deleted the tabulator after the speaker code in the first utterance of the file called 02.cha, and inserted a tabulator into the second one. That is, we changed the string "the father had always prevented them" to "the father had always → prevented them" and ran the command "check 01.cha". This elicited the following output from CLAN:

```
From file <02.cha>
```

```
*** File "02.cha": line 8.
*XYZ: In her newspaper column Error, Stage left, Anna
Quindlen is accusing Actors' Equity
Use TAB character instead of space character(s).(4)
*** File "02.cha": line 11.
*XYZ: → she says that → Actors' Equity likes it
Illegal character(s) '<tab>' found.(48)

Warning: BASIC SYNTAX ERROR - Second pass not attempted !
Warning: Please repeat CHECK until no error messages are
reported!
```

Because the second stage of the checking process was not performed, it is quite possible that the file 02.cha contains many further errors, which the program is not detecting at this point, and which will only become apparent once you have corrected the intial oversights and run the check command again

The file 01.cha, on the other hand, does not contain any such basic syntax errors, but will produce a number of other error messages. In each case, the error message is preceded by the utterance which contains the violation, with the offending bit underlined. The first two error messages we received pertained to the fact that a number was used within the text:

```
*** File "01.cha": line 6.
*XYZ: the story is about the relationship between a father
and his 2 daughters and about the way he influenced them.
Numbers are not allowed inside words.(47)
```

```
*** File "01.cha": line 6.
*XYZ: the story is about the relationship between a father
and his 2 daughters and about the way he influenced them.
Numbers should be written out in words.(38)
```

As you can see, you may sometimes get multiple error messages triggered by the same violation, so replacing "2" with "two" here will fix both these error messages.

In the next utterance there are two sentences, each of which elicits an error message. The first one pertains to a new utterance (starting with a capital letter) and the second one to an utterance that does not have an utterance delimiter (a period) at the end:

```
*** File "01.cha": line 7.
*XYZ:  after their father's death, the two grown-up sisters,
Constantia and Josephine weren't controlled by their father
anymore. They also couldn't rely on him anymore.
Utterance delimiter must be at the end of the
utterance.(36)
*** File "01.cha": line 11.
*XYZ:  the story ends with a dialogue between the two women
about the question whether they should keep Kate as a maid
or fire her
Utterance delimiter expected.(21)
```

Again, these problems could easily be fixed by dividing the first utterance into two utterances (and putting *XYX: → in front of the second utterance) and adding a period at the end of the last utterance and then running the CHECK command again.

In order to keep errors to a minimum, it is important to stick to the guidelines above. The most common errors in CHAT files are:

a. utterance delimiters (punctuation marks such as ., ?, !) within the utterance
b. no utterance delimiter mark at the end of utterance lines
c. the obligatory headers are not all there, or not in the correct order
d. upper case letters within words
e. numbers in the text (they have to be spelled out)
f. two tiers of the same kind (e.g. two error tiers) are associated with one utterance tier
g. a comma is immediately followed by the next word (there has to be a space between the two)
h. a code starting with & (e.g. &=laugh) is immediately followed by a punctuation mark (there has to be a space between the two)
i. there is an empty line after the @End line
j. there is an empty line before the @Begin line
k. there are empty lines in the text

Creating your own set of tools to code data

When you are transcribing and coding data in CHAT format, there is a relatively limited set of coding operations which you have to perform multiple times in each file. For example, let us assume that you want to code two dependent tiers: an %xsen tier, where each sentence will be coded as simple, compound or complex and an %xcla tier, where you classify the clauses as nominal, adverbial, relative, non-finite or other.

There are two options for performing such coding operations. The first is to transcribe and code your data directly in CLAN, that is, to use the program as your text editor. In this case, you can create your own coding files and enter the dependent tiers in the Coder mode. The steps which are necessary to do this are described in the CLAN manual. As was mentioned above, doing this can be more convenient, as it will help you prevent all kinds of problems that may be created by converting text files created with other text editors to text-only format.

The second option is to use Word as your text editor (saving your files in text only format) and create your own set of tools with the help of this program. While this may lead to a set of its own problems, the advantage is that with the help of these tools you can automatize many of the coding procedures. Which of these two options you decide to use is a matter of personal preference. As the CLAN program and the Coder mode are described in some detail in the CLAN manual, we will focus here on how to create tools with the help of Word.

Recording Macros in Microsoft Word

Manually inserting the codes necessary for CLAN analyses can be a laborious task, and there is always the danger of miss-typing codes, which will then not be counted correctly by the computerized analyses. It is therefore both more convenient and more reliable to have the computer do the coding work for you, wherever possible. In the CLAN coder mode, you can define a set of options for each dependent tier, thereby ensuring that only the correct codes will be chosen. In the present section we will explain how you can automatize different steps of the coding procedure in a text editor program, such as Microsoft Word.

The Word program allows you to record a string of operations as a kind of mini-program, which is referred to as a 'Macro'. In other words, a Macro is basically a “recording” of every

key stroke, cursor movement or mouse click which you have performed. Macros are very useful for operations you need to perform often in exactly the same kind of way. For example, in our data we want to code the proportion of finite verbs in relation to the total number of words in each text in order to get an impression of sentence length and complexity. In order to do this, we have to identify each finite verb manually in the text (unfortunately, most such identification tasks can not yet be performed reliably by computers) and then add a code after the verb. This code has to be something which CLAN will recognize as not belonging to the actual utterance, for example "[% f]". Typing this string out every time is cumbersome: first, we have to move the cursor to the correct place (which we will most likely do by means of a mouse click) and then we have to type the angled bracket, the percentage sign followed by a space and the letter 'f', and the other angled bracket. It is also easy to make a mistake in typing this string. It would therefore be more convenient and reliable to either have a button that we could click on or a keyboard shortcut which will insert the code at the location of the cursor.

We can have this done automatically by recording a Macro. You can find the “how to” on the CD.

After you have recorded the macro and you move the cursor to a different place (the space after the next finite verb) in the text and press the keyboard shortcut you have chosen, the same code will appear.

Macros can be run three different ways: from the menu Tools-Macro, with a keyboard shortcut (as we just discussed), or from a Macro toolbar. A toolbar is especially handy if you have to code texts for many different things. Note, however, that the possibilities of creating such toolbars has been severely restricted in the 2007 version of the Microsoft Office package, so if you are using this version of the program, you will have to rely on the Macro menu or keyboard shortcuts.

Creating a toolbar for macros

Let us assume that you need to create several macros to code your texts. You now want to create a toolbar from which you can pick your macros as required, by simply clicking on them. Such a toolbar can be created in the menu Tools - Customize. Pick the tab 'Toolbars' and click 'New'. Give a name to your new toolbar, such as "Coding". A small new toolbar will appear on your screen (which, for the moment, is empty). Pick the tab "Commands". In the window on the left, you can see the different menus. Scroll down until you get to "Macros" and click on this. You now see all the Macros you have created in the right hand window, and you can grab them and drag them to the new toolbar. (Note that the "Command" window also contains the option "Keyboard", where you can change or assign keyboard shortcuts to your Macros - or to any other Word function). The names for the Macros that appear on this new toolbar are very long, as they contain the entire identification string (e.g. "Normal.NewMacros.finiteverb), but while the "Customize" window is still open, you can right-click on the name and change it to something shorter and more convenient.

Once you familiarize yourself with the steps associated with creating Macros, you will find them a fantastically useful tool. They can be used, for example, to automatically reformat the output from programs such as CLAN or SPSS, to reformat paragraphs, tables or citations from one style to the other, or to insert the CHAT codes you want.

On the CD, we give step-by-step instructions how to create Macros to mark a word as a code-switch, mark a stretch of text as an error, insert a dependent tier of the type %err after the utterance containing the error, and insert an %xsen tier of one of three types (simple, compound, complex) after an utterance. We also show how you can work even more efficiently by copying and pasting the program text for macros that you already have created. We hope that the examples of how to record Macros and design Toolbars will allow you to

develop your own tools for the specific coding options you require. For example, you could write a Macro to insert the obligatory headers in the file, to code retractions and repetitions (similarly to the Error Macro described on the CD) and so on. Once you have the set of tools you need, you can assign keyboard shortcuts to each Macro (using the Menu "Tools - Customize - Keyboard", if you have not already assigned them in the process of originally creating the Macro) and/or place them on a new Toolbar, called, for example, "CHATTools" (or, of course, on an existing one). Once all of your files have been sufficiently coded and checked in CLAN, you are ready to begin analyzing them. In the following, we will explain how to obtain the measures that we discussed in Table 1.

Using CLAN to perform an analysis of lexical frequency and a morphological analysis

As was pointed out above, the most straightforward way to measure whether there is an increase in lexical diversity in language development is by investigating the *Type-Token-Ratio* (TTR). TTRs are based on a stretch of discourse, and give the proportion of tokens to types (types divided by tokens). Types are the number of *different* words and tokens are the *total* number of words. CLAN can calculate this for you by means of the command used to perform a frequency analysis. You can see how this command works by changing your working folder to "initial" on the CD, and then at the command prompt, typing the command:

```
freq 01.cha
```

This command should produce a list of items which begins with the proper nouns used in the file, and then lists all other words in alphabetical order, together with the number of times they have been used:

```
> freq 01.cha
freq 01.cha
Wed Sep 16 11:56:55 2009
freq (31-Jul-2009) is conducting analyses on:
  ALL speaker tiers
****************************************
From file <01.cha>
  2 Constantia
  1 Josephine
  1 Kate
  4 a
  5 about
  2 after
  1 also
```

The list runs through the alphabetical list of words used in the text, until you reach the last ones. Below this is the number of types and tokens used in the text, as well as the type-token ratio.

```
  1 very
  3 way
  1 were
  3 whether
  1 which
```

```
  1 with
  2 women
  1 wondered
  1 would
------------------------------
  111  Total number of different word types used
  207  Total number of words (tokens)
0.536  Type/Token ratio
```

As you see, the word "were", which of course is a form of "to be", is counted as a separate item here (and if you scroll up, you will find other forms of the verb "to be", such as "is"). This problem is addressed on the CD when we discuss the morphological analysis.

On the basis of this command, it is possible to calculate the simple TTR for any text which has been coded in CHAT format. However, as was pointed out above, TTRs are not very reliable, in particular for texts of different length. CLAN therefore also offers the possibility of calculating the measure D (see McKee, et al. 2000). The command associated with this measure is

```
vocd 01.cha
```

The output from this command is fairly long and complex - if you are going to work with this measure, we suggest you familiarize yourself with the underlying theoretical and mathematical assumptions, which will help you understand how D is measured and what it implies. For the purposes of the present analysis, however, you can find the D measure for each text at the very end of the output, under "D_optimum_average". Note that D is based on

random sampling and will therefore be slightly different every time you run the command, even on the same material - running it three times consecutively on '01.cha', for example, returned a D of 62.65, 62.31 and 62.87, respectively.

In the previous two examples, we have run the freq and the vocd command on only one file. The folder "initial" contains a total of 22 texts, which were produced by an advanced language learner over a period of 3 years. The higher the text number, the later in the developmental period the text was produced (that is, the more advanced and sophisticated we would expect the text to be). In order to see how type-token ratios and D developed over this period, we can run the above commands on all texts included in the folder in one go. This is done by the command

```
freq *.cha
```

or

```
vocd *.cha
```

The * in this command is a wildcard: "*.cha" just means "any file which ends in .cha, irrespective of the filename". Typing in this command therefore produces the same type of output you saw above for each of the texts in the folder.

If you now scroll up through the output, you will see that not all files are listed here. The CLAN output window is limited as to how much text it can display, so if the output gets too long, the beginning disappears. In commands which affect a large number of files, you can direct the output to a new file by typing

```
freq *.cha > freq.txt
```

This command creates a new file, called freq.txt, in which the output will be placed. Apart from the calculations for the TTR for each text, this file now also contains a full list of every word used in every one of the texts, together with an indication of how often each word appeared.

You can use the information in this list for investigations into lexical diversity which go beyond blind frequency-based analyses. For example, Laufer and Nation (1995, 1999) propose that the "Lexical Frequency Profile" (LFP) of a learner should be based on analyses which measure how many of the words used belong to the 1000 most frequent words in the language, how many to the next 1000, and so on, taken in conjunction with counts of 'rare' or 'advanced' words, and words which belong to specific registers, such as academic writing. Such analyses can indeed provide insight into vocabulary knowledge, but they require that word frequency lists are available for the language under investigation, and they are also very time-consuming to conduct.

We therefore propose that investigations of relatively large learner corpora may adopt a less costly approach by using their own corpus as the basis for a frequency count. Based on analyses such as the one illustrated above with the help of CLAN, it is easy to calculate measures such as what proportion of the words a learner has used are among the 50 most frequent items in the overall corpus, what proportion of words are unique to that particular learner, what is the average overall frequency of all the words used by a particular learner, and so on. On the CD we demonstrate step by step how to do such an analysis. We also show in detail how to conduct a morphological analysis.

Limiting frequency counts

In some cases, you may not want a frequency count to be performed on *all* of the items on a specific tier. For example, recall that we used the code [% f] to mark every finite verb. You may want to limit the frequency count to that particular string, in order to see how many finite verbs occur in each text. This is simply done by adding the code " +s" to the frequency command, followed by the text which you want to include, in this case

```
freq *.cha +s"[% f]"
```

For this option, you can use an asterisk as a so-called wildcard, for example, the command

```
freq *.cha +s"a*"
```

would produce a list of only words which start with the letter *a*, and the command

```
freq *.cha -s"*@s"
```

will exclude any word with the suffix "@s" (that is, all words that have been marked as code-switches).

Analyzing dependent tiers

As you have seen above, the frequency analysis that can be performed by CLAN can be limited to dependent tiers, such as the morphologically tagged lines which start with %mor. This function can be exploited for other analyses as well. Recall that above we discussed inserting a tier called %xsen to code different types of sentence (simple, compound, complex). On your CD, you will find a folder called "alltiers". This folder contains the same

22 texts, but for each utterance, we have coded such an %xsen tier, and a tier which we called %xcla, which contains information on the type of clause contained in the utterance. Both tiers have four possible values:

%xsen: simple, compound, complex, cmpdcpx
(this last code refers to "compound-complex" sentences
%xcla: adverbial, nominal, nonfinite, relative

You can now check how often each sentence and clause type was used in each of our 22 files by typing:

freq -t* +t%xsen *.cex
or
freq -t* +t%xcla *.cex

The output from this command will tell you how many instances of each sentence and clause type have appeared in each of the texts. You can also direct the output to a file, e.g. "cla.txt" by adding " > cla.txt" at the end of the command.

Visualizing the data

With the help of the functions described above, you can count a large number of lexical and grammatical variables in your text, and import those to your spreadsheet or statistics software. On your CD, you will find an Excel-file named "Analysis.xls" which contains all of the

variables which we have described above. The file contains two data sheets, the first one is called "Data". In this sheet, the output from the CLAN analyses have been compiled.

a. Columns B through G contain the results of the various analyses of lexical diversity described above. B, C and D are the output from the simple Type-Token Analysis, E contains the measure D (the output from the vocd command). In column F you find the percentage of content words used in each text which belong to the 50 most frequent lexical items in the overall corpus, G contains the percentage of content words which are used only once in the entire corpus, and H gives the average length of all content words used for each text.
b. In column I, you can find the number of finite verbs used in each text. When you are analyzing texts of varying length, such absolute numbers are, of course, not very illuminating (a longer text will by definition contain more items of a certain type). Therefore, column J is a recalculation: the number of finite verbs (the value in column J) is here divided by the number of tokens. If the value in this column is, for example, 10, this means that every 10th word in the text was a finite verb.
c. The next four columns contain the outcome of the frequency analysis on the %xsen tier. Each value tells you what percentage of the utterances in that text were of the type compound/complex, compound, complex or simple.
d. The next four columns give you that same information with respect to the %xcla-tier.

Our main research question from a DST perspective is how development took place: Which variables seem to develop at the same time (connected growers), which variables seem to compete (competitors), what variables seem to have developed prior to other ones (precursors)? Our assumptions and questions will be as follows:

Coding and extracting data

1. Sentence constructions will go from simple to more complex.
2. How does complexity emerge?
3. Are there any competing variables?
4. Lexical items will go from more frequent to less frequent.
5. Is it true that the proportion of highly frequent words decreases as the writer becomes more proficient?
6. Is the increase in vocabulary size as the writer becomes more proficient reflected in an increase in TTR-based measures, such as D?
7. Does the proportion of unique words increase as the writer becomes more proficient?
8. Does average word length say something about lexical development?
9. There may be some kind of interaction between the lexicon and sentence complexity.
10. Is there any particular relation between a general complexity measure (such as number of finite verbs per 100 words) and a general lexical measure (such as the use of frequent words or average word length)?

Below, we will briefly show how the data pertaining to these questions can be explored by means of visualizations. In the next chapter, we will look at statistical procedures which can establish how likely it is that any patterns we may find are merely coincidental, or whether there is a high probability that they are actually indications of change through development.

The development of complexity in syntactic constructions

Our assumption above was that the proportion of simple sentence constructions would decrease and a higher proportion of more complex constructions would be visible in the course of language development. As simple sentences and compound sentences consist of main clauses only, they are simpler than complex or compound-complex sentences, which

have finite dependent clauses. Even more complex than finite dependent clauses are non-finite constructions such as ***to induce*** *their incidental attention to some specific linguistic forms when* ***processing*** *either input or output.* An increase in complexity may furthermore be indicated by longer noun phrases as in *one of the major purposes of these tasks for learners*. To test our assumption that sentence constructions develop from more simple to more complex, we can look at the development of the following variables:

a. proportion of simple and compound sentences
b. proportion of complex and compound-complex sentences
c. the number of words per finite verbs (total number of words divided by the number of finite words)

Figure 3 shows how these three variables develop over time. We have also added a polynomial trend line to the finite verb ratio (FV) line to see what the general trend is.

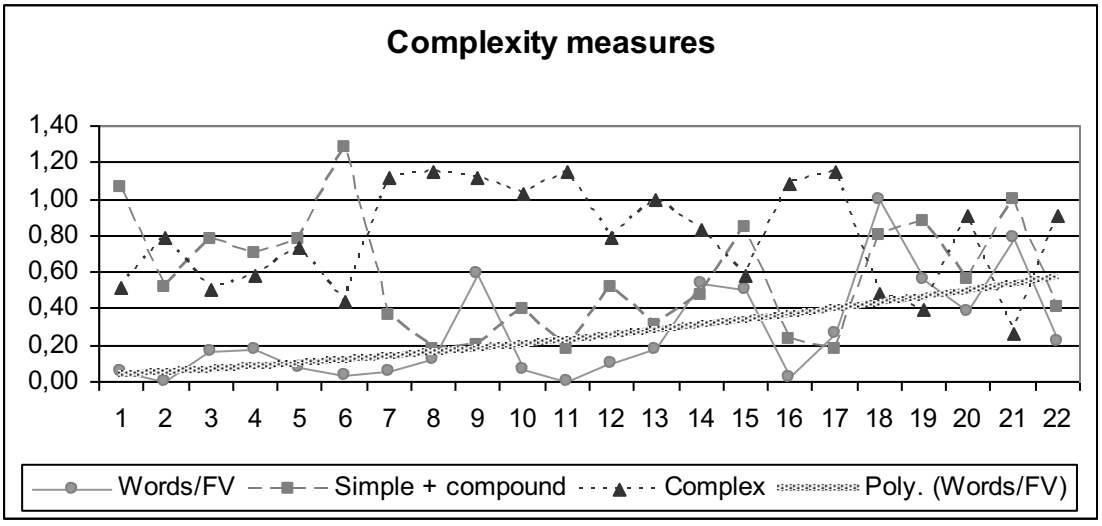

Figure 3. Development of sentence complexity measures

Figure 3 shows that up to Text 5, the writer used both simple and complex constructions to a similar degree. After this point, a marked dip of simple and compound sentences occurs,

accompanied by an increase in complex sentences. Meanwhile the Words to Finite Verb ratio remained relatively low except for a few small peaks. This means that sentences were made more complex by means of finite dependent clauses. After Text 16 the Words to Finite Verb ratio increases perceptibly, while at the same time the distribution of simple/compound and complex constructions evens out again. This indicates that at that stage, non-finite constructions and longer noun phrases contribute to overall complexity to the same degree as dependent clauses. We may conclude that our writer moved from simpler constructions to more complex ones.

One of the questions is how this complexity emerged. Did all dependent clauses develop at the same time? Did some occur before others? Are there some constructions that compete with each other? To find an answer to these questions, we plotted the types of finite dependent clauses (adverbial, nominal, relative) and non-finite clauses (not separated for different kinds), first separately and then together in different combinations. Here we will show the development of the proportion of adverbial and relative clauses, each with a polynomial trend line (Figure 4).

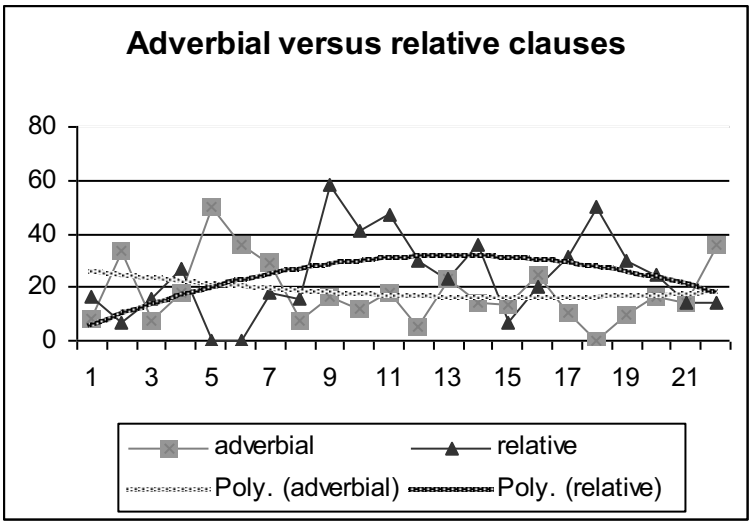

Figure 4. The development of adverbial and relative clauses, each with a polynomial trendline.

Figure 4 shows that the number of adverbial clauses first increased somewhat and then decreased and leveled off until the last few texts. Relative clauses, on the other hand, increased at the same time as adverbial clauses decreased. The trend line seems to show typical U-shaped behavior, which means that as person is trying to learn something s/he may overuse it for a while. When we looked at the development of nominal clauses versus relative clauses, a very similar pattern emerged. These patterns suggest that relative clauses develop somewhat later than adverbial and nominal clauses, and at one point seem to compete with them. Another observation is that the adverbial clauses and relative clauses compete. If one is used, the other is not, and vice versa. We will explain how to test this assumption in the next chapter.

Lexical items will go from more frequent to less frequent

As far as vocabulary is concerned, one would expect a beginning learner to use highly frequent items and then proceed to acquire less frequently used words as s/he becomes more proficient. We would also expect the use of vocabulary to become more creative (with less repetition of the same items) and more sophisticated. To test these assumptions we calculated the lexical diversity measure D, the frequent word measure (for each text, we calculated the percentage of the total content words belonging to the 50 most frequent items in the overall corpus) and the unique word measure (for each text, the percentage of words which were only used once in the entire corpus). A final measure of lexical sophistication is the average length of content words in each text. We would expect D, average word length and unique words to

increase and the percentage of frequent words to decrease. Figure 5 shows the development of the frequency measures.

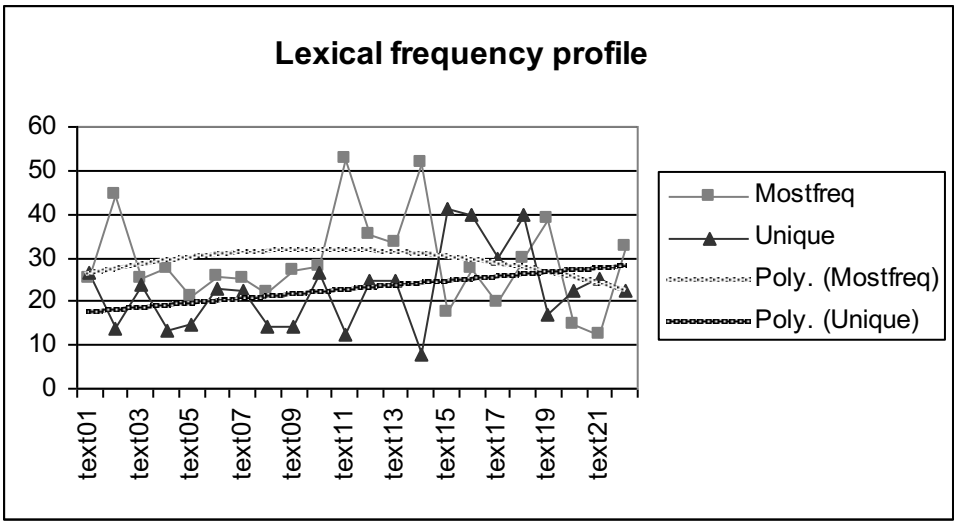

Figure 5: Development of lexical frequency profile

It is evident from Figure 5 that our assumptions concerning lexical frequency are met. The number of unique words increase and the number of very frequent words decrease. After text 15, there seems to be a balance between the two measures. Figure 6 then shows the development of D, the proportion of unique lexical items, and average word length.

These three measures illustrate the difficulty of visually representing measures on different scales: in our data, D has a maximum value of 104 and a minimum of 44, the proportion of unique words varies between 8 and 41%, and the words are between 4 and 8 letters long. Obviously, these three measures cannot be represented together in a single graph. The Excel-file Analysis.xls therefore contains a second worksheet, called 'Data recalculated 0 to 1'. In this sheet, all the values from the sheet 'Data' are proportionally recalculated to a scale

with a maximum of 1 and a minimum of 0.[2] The procedure performed in this recalculation is illustrated on the CD.

This then allows us to represent the development of these three variables within the same graph.

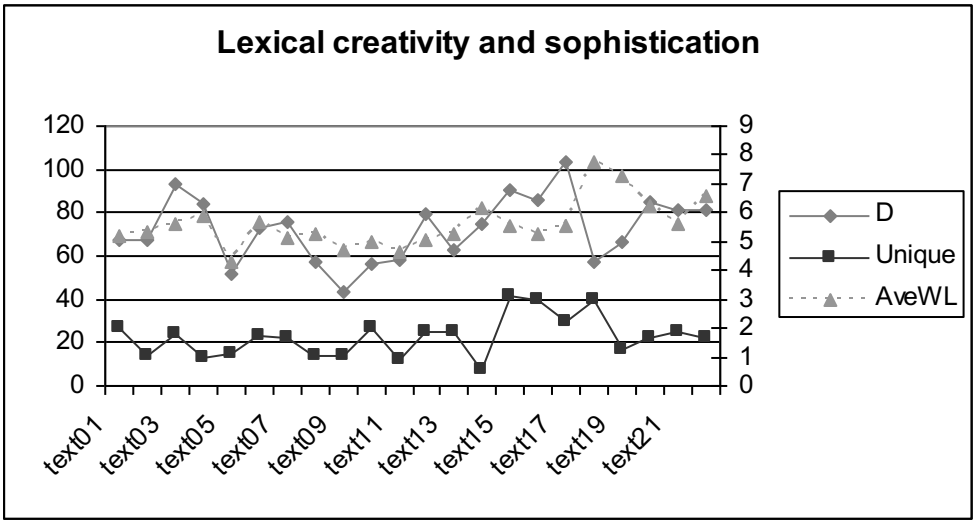

Figure 6. Development of lexical diversity and sophistication

Our assumption that lexical diversity (D measure) and sophistication (unique words and average word length) increase over time is met. But do they measure the same things? Even though one would expect unique words and lexcial diverstiy (D) to correlate strongly because a higher number of unique words and D should essentially be the same thing, the correlation between them is not very strong (0,38) and their relationship changes over time, especially around text 19. Also average word lenth, a very general measure of lexical sophistication does not measure exactly the same thing as lexical diversity because the correlation is even lower (0,20) and the measures diverge after text 15. We may conclude that these measures tap

[2] This recalculation can be performed by means of a simple formula. If you take a value X in a scale Y, you first need to establish the scale maximum (the highest value in your scale) and the minimum (the lowest value in the scale). The formula to recalculate X is then

$$\frac{X - \text{Scale Minimum}}{\text{Scale Maximum} - \text{Scale Minimum}}$$

If we want, for example, to recalculate our D values, the scale maximum (in text 17) is 103.78, and the minimum (in text 09) is 43.60. To recalculate a value X, for example the D value of text 06 (73.06), we insert these numbers into the above formula: (73.06-43.60)/(103.78-43.60) = 29.46/60.18 = 0.49

into slightly different aspects of lexical development. In the next chapter we will explain how to measure whether these patterns are meaningful or random.

Another question we had was if the TTR and D would show the same picture and as Figure 7 shows, they do (again, the variables were recalculated to a scale between 0 and 1). The two measures show a strong correlation (0,83). (Note that for the present corpus, this is not a surprising finding, since all our texts are of the same length and relatively short.

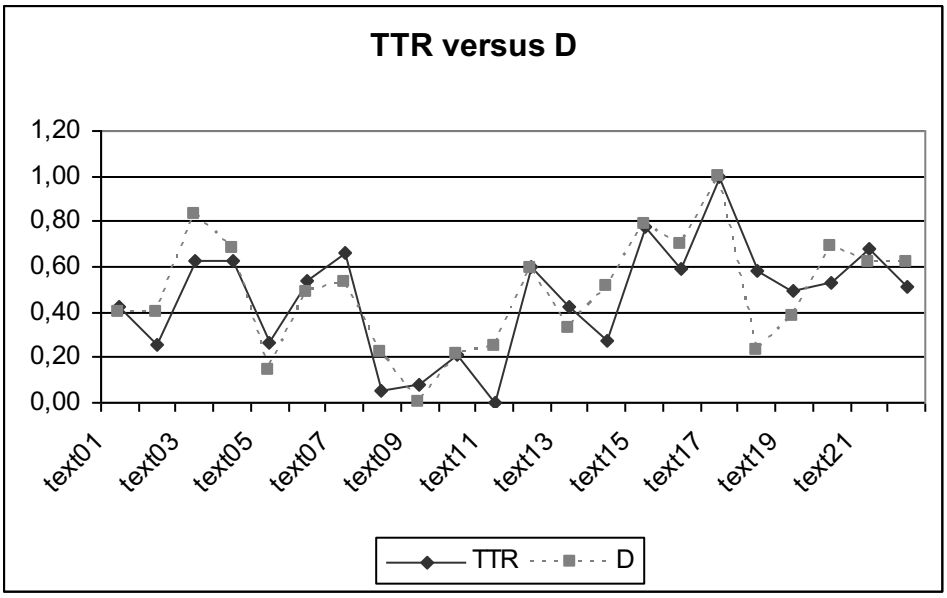

Figure 7. Development of TTR versus D

Interaction between sentence complexity and lexical sophistication

Finally we wanted to establish whether there is interaction between lexical sophistication and sentence complexity. It is possible that these two measures develop hand-in-hand, because if you use more complex sentences, you are likely to use less frequent and more sophisticated words like *induce* and these words in turn occur with more technical language, which is usually more complex. On the other hand, it is possible that a more advanced vocabulary (comprising less frequent words) has to have been acquired before more complex language can be used, and in that case vocabulary would be a precursor for sentence complexity. This is a reasonable assumption for L1 acquisition as well as early L2 learning, but we do not know

how these variables might interact in data from a more proficient learner. Another possibility is that a learner may have trouble focusing on two different aspects of language at the same time, because both require cognitive resources. To discover the relation between lexical sophistication and sentence complexity we looked at the most general complexity measure, the average distance between finite verbs and the ratio of unique words.

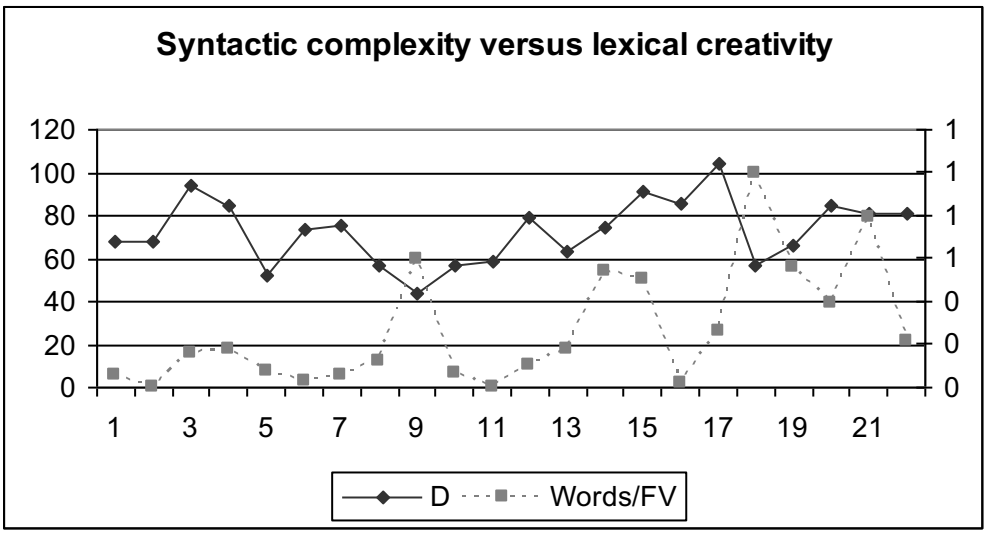

Figure 8. The interaction between sentence complexity and lexical diversity

Figure 8 shows that after Text 5, the two variables seem to compete quite strongly until Text 19 where they seem to go hand in hand again. Both variables furthermore appear to show a dip in text 09, which raises the intriguing possibility that at that point, some other variable was in a state of strong development. We may conclude that for a part of the developmental process these two measures compete. Again, in the next Chapter we will discuss how to measure whether these observations are meaningful or not.

Conclusion

At the beginning of this chapter we argued that for a usage-based DST perspective on second language development, it is best to use longitudinal, naturalistic data (spoken or written) and to look at how many different linguistic variables develop over time. The reason is that we do not only want to see if they develop, but also when they emerge, how they develop, and how different variables may interact.

The assumption underlying the DST interpretation that there can be a trade-off between different skills in SLD is that the learner has to make some kind of choice as to which component of the message s/he will allocate a larger proportion of the finite cognitive resources to. Such tradeoffs will, by necessity, be less pronounced in written texts than in spoken data: The written data we used were produced without time limit, and presumably re-written several times in the process. During such revisions, the author had the option to focus on different components of her linguistic repertoire each time she read and changed the text. Nevertheless, the linguistic subcomponents clearly emerge as competing with each other, in particular at certain moments in her development. It is therefore reasonable to assume that such effects would be more pronounced in a longitudinal corpus of spoken learner data.

The main purpose of this chapter was to show you how you can use dedicated linguistic software, but also widely available text editing and spreadsheet software, specifically CLAN, Word and Excel, to code a range of different variables consistently and efficiently. Lastly, we showed how the data may be explored through charts, and which observations and assumptions could be made on the basis of this visual presentation of the data. Our main concern was not so much to show specifically how we coded and analyzed the data at hand, but to provide you with templates which you might apply and use in the exploration of your own data.

In the next chapter, we will focus on how to interpret longitudinal case data and how to test whether the patterns found are meaningful or random.