The CHILDES Project

Tools for Analyzing Talk – Electronic Edition
Volume 2: Transcription Format and Programs

Part 2: The CLAN Programs

Brian MacWhinney
Carnegie Mellon University

March 29, 2010

Citation for last printed version:

MacWhinney, B. (2000). The CHILDES Project: Tools for Analyzing Talk. 3rd Edition. Mahwah, NJ: Lawrence Erlbaum Associates

---

1 Getting Started

This manual describes the use of the CLAN program. The acronym CLAN stands for Computerized Language ANalysis. It is a program that is designed specifically to analyze data transcribed in the format of the Child Language Data Exchange System (CHILDES). Leonid Spektor at Carnegie Mellon University wrote CLAN and continues to develop it. The current version uses a graphic user interface and runs on both Macintosh and Windows machines. Earlier versions also ran on DOS and Unix without a graphic user interface. CLAN allows you to perform a large number of automatic analyses on transcript data. The analyses include frequency counts, word searches, co-occurrence analyses, MLU counts, interactional analyses, text changes, and morphosyntactic analysis.

Chapter 1 explains how to install and learn CLAN. Chapter 2 provides a tutorial on how to begin using CLAN. Chapter 3 explains how to use the editor. Chapter 4 explains some additional features, how to access help, and how to report bugs. Chapter 5 provides detailed descriptions of each of the CLAN commands. Chapter 6 provides details regarding particular command options. Chapter 7 gives some exercises for learning CLAN.

1.1 Learning CLAN

In order to learn CLAN, you will want to first work through the Basic Tutorial in Chapter 2. That tutorial will give you a basic understanding of the program. After going through these initial steps, you will want to explore the features of the editor by working through chapter 3 on the editor. Then you will want to learn each of the various analytic commands, concentrating first on the five basic commands illustrated in the tutorial. Then you should move on to the Advanced Tutorial in Chapter 4.

1.2 Installing CLAN

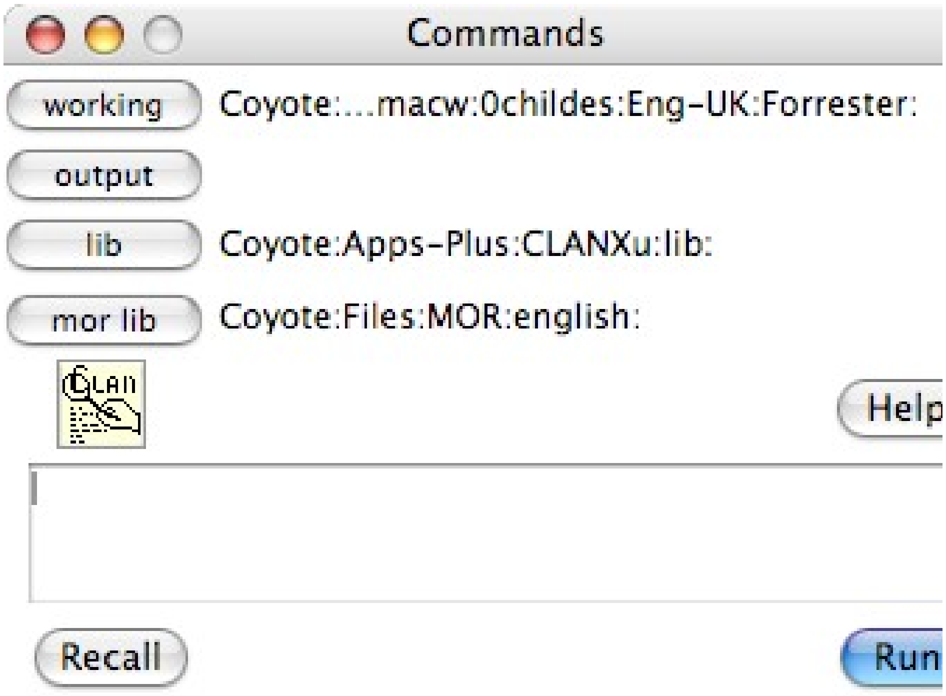
CLAN can be retrieved from http://childes.psy.cmu.edu. Macintosh CLAN is distributed in a StuffIt file with the extension .sit. You will need to have a copy of StuffIt Expander™ to expand it. You drop this file onto StuffIt Expander and it will expand. The CD-ROM version of Macintosh CLAN can be dragged onto your desktop. Macintosh CLAN requires OS X and is not compatible with Classic or OS 9.

Windows CLAN is distributed in a file called clanwinu.exe. You can copy this file to your hard drive from either the Internet or the CD-ROM. You then click on the file and it will run InstallShield that then installs CLAN in c:\childes\clan. If you already have a copy of CLAN installed, you will need to first remove it using Add/Remove Programs. Windows CLAN is not compatible with Windows 95/98/ME.

CLAN is also available for Unix. However, the Unix version of CLAN only provides the CLAN commands and does not provide the CLAN editor.

2 Tutorial

Once you have installed CLAN in accord with the instructions in the previous chapter, you start it by double-clicking on its icon. After this, a window titled **Commands** opens up and you can type commands into this window. If the window does not open automatically, then just type *Control-d* (Windows) or ⌘-*d* (Macintosh). Here is what the **Commands** window looks like on Macintosh. The Windows version is similar.



Coyote:...macw:0childes:Eng-UK:Forrester:

Coyote:Apps-Plus:CLANXu:lib:

Coyote:Files:MOR:english:

2.1 The Commands Window

This window controls many of the functions of CLAN. It remains active in the background until the program is terminated. The main components of the **Commands** window are the command box in the center and the several buttons. The first thing you need to do when running CLAN is to set the **working** and **lib** directories.

Setting the Working Directory

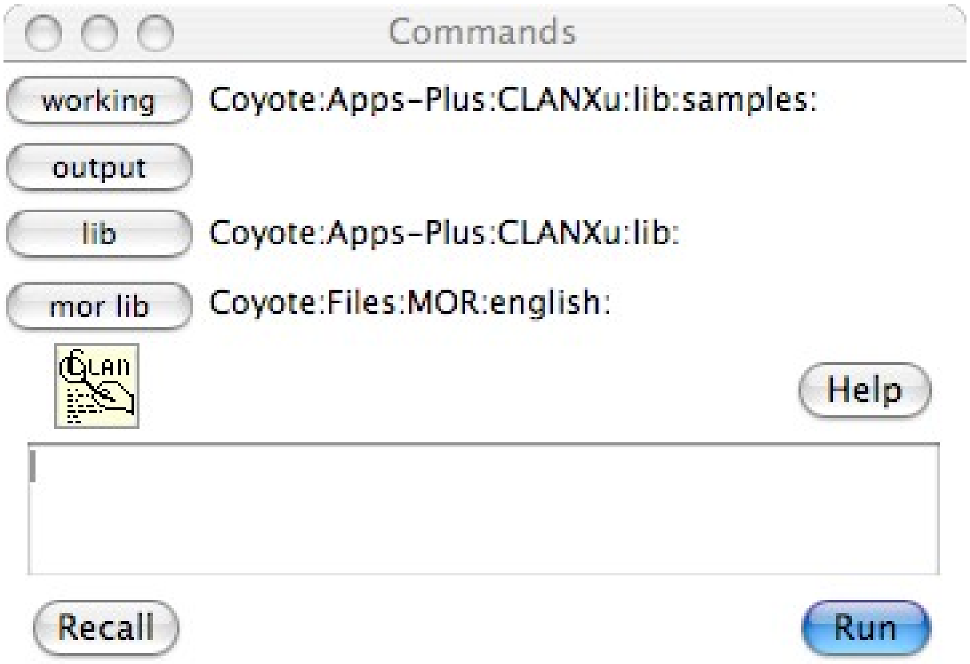
The working directory is the place where the files you would like to work with are located. For this tutorial, we will use the CLAN library directory as both our Working directory and our Library directory. To set the working directory:

1. Press the **working** button in the Command window (see screen grab above).
2. Locate the directory that contains the desired files. For this tutorial, please use the **lib/samples** directory inside the CLAN directory.

3. Press the **Select Current Directory** button (see next screen image).

After selecting the current directory, you will automatically return to the **Commands** window. The directory you selected will be listed to the right of the **working** button. This is useful because you will always know what directory you are working in without having to leave the **Commands** window.

After you have set the working directory, go through the same process to set the library directory. You do not need to worry about setting the output directory. By default, it is the same as the working directory. Also, for now, you do not need to worry about the "mor lib". Once you have set your working and lib directories, your commands windows should look more or less like the picture below. Of course the actual paths will be different on the top levels, but the bottom levels should say "samples" and "lib".



working Coyote:Apps-Plus:CLANXu:lib:samples:

lib Coyote:Apps-Plus:CLANXu:lib:

mor lib Coyote:Files:MOR:english:

Now, to test out your installation, type the command "freq sample.cha" into the Commands window. Then either hit the return key or press the **Run** button. You should get the following output:

```
> freq sample.cha
freq sample.cha
Tue Aug  7 15:51:12 2007
freq (03-Aug-2007) is conducting analyses on:
  ALL speaker tiers
****************************************
From file <sample.cha>
  1 a
  1 any
  1 are
  5 chalk
  1 delicious
  1 don't
  1 eat
  1 good
  1 hey
  1 i
  1 is
  1 it's
  1 mommy
  1 more
  2 neat
  1 nicky
  1 not
  2 oh
  1 other
  3 see
  1 some
  2 that
  1 that's
  2 there
  3 to
  3 toys
  3 want
  1 what
  2 what's
  1 wonderful
  2 yeah
  2 you
------------------------------
   32  Total number of different word types used
   51  Total number of words (tokens)
0.627  Type/Token ratio
```

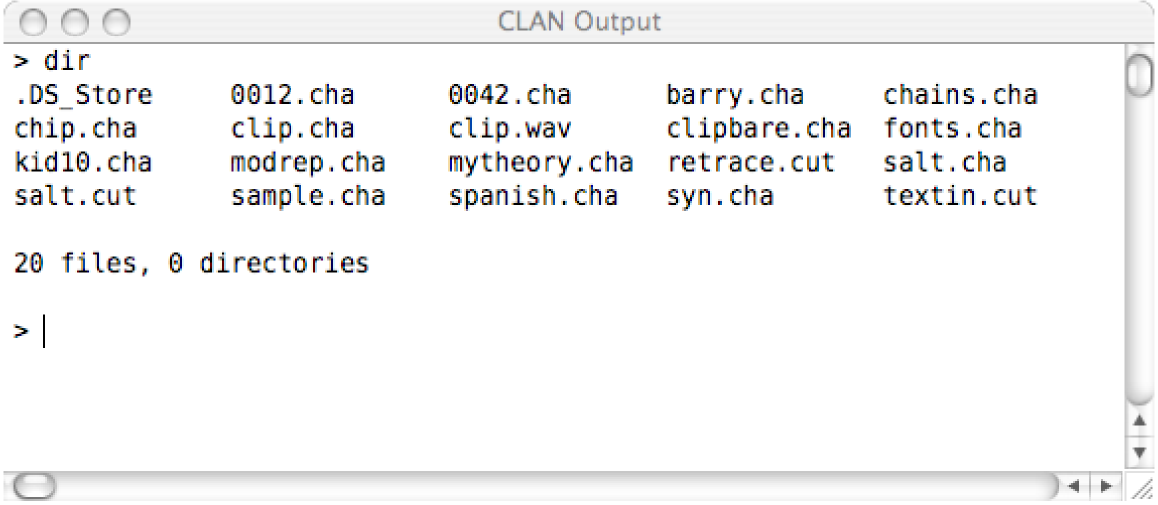
The output continues down the page. The exact shape of this window will depend on how you have sized it.

The Recall Button

If you want to see some of your old commands, you can use the recall function. Just hit the **Recall** button and you will get a window of old commands. The **Recall** window contains a list of the last 20 commands entered into the **Commands** window. These commands can be automatically entered into the **Commands** window by double-clicking on the line. This is particularly useful for repetitive tasks and tracking command strings. Another way to access previously used commands is by using the ↑ arrow on the keyboard. This will enter the previous command into the **Commands** window each time the key is pressed.

The HELP Button

The **Help** button can also give you some basic information about file and directory commands that you may find useful. You enter these commands into the command box. To test these out, just try typing *dir* into the **Commands** window. You should get something like this in the **CLAN Output** window:



```
> dir
.DS_Store     0012.cha      0042.cha      barry.cha     chains.cha
chip.cha      clip.cha      clip.wav      clipbare.cha  fonts.cha
kid10.cha     modrep.cha    mytheory.cha  retrace.cut   salt.cha
salt.cut      sample.cha    spanish.cha   syn.cha       textin.cut

20 files, 0 directories

> |
```

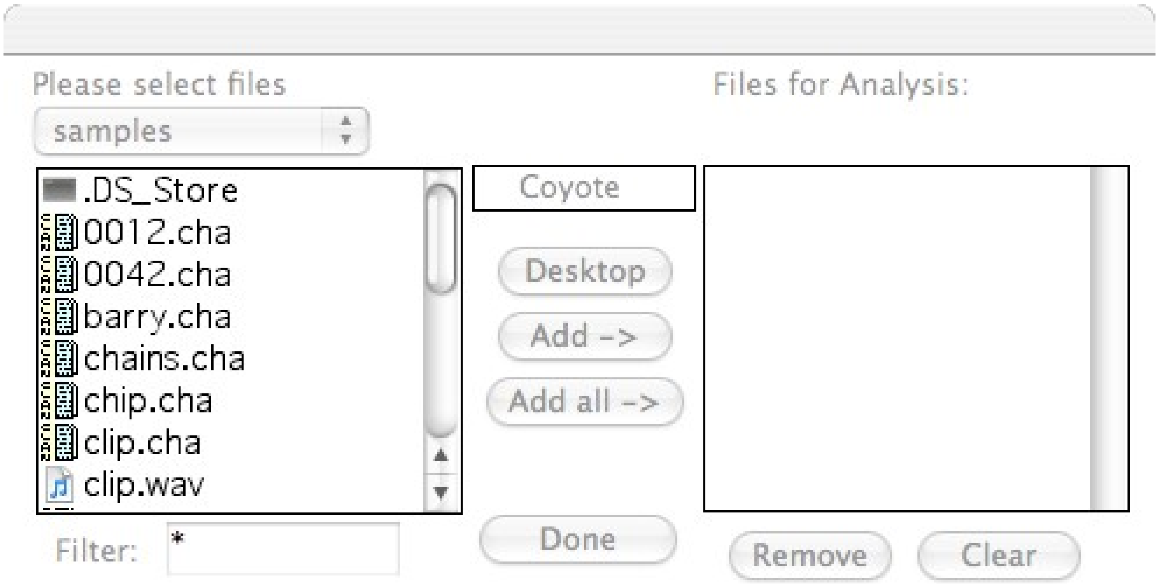
You may want to resize this window if text is being cut off.

The CLAN Button

The CLAN button gives you a list of CLAN analytic commands you can run. If you already know which command you want to run, you may find it faster just to type the name in the **Commands** window. However, just for practice, try clicking this button and then selecting the FREQ command. The name of the command will then be inserted into the **Commands** window.

The File In Button

Once you have selected the FREQ command, you now see that the **Files In** button will be available. Click on this button and you will get a dialog that asks you to locate some input files in your working directory. It should look like this the screenshot on the next page. Scroll down to the file sample.cha and double-click on it to move it to the right. The files on the right will be the ones used for analysis. The **Remove** button that appears under the **Files for Analysis** scrolling list is used to eliminate files from the selected data set. The **Clear** button removes all the files you have added. The **Filter** text box shows the file extension of the selected data set. Those files with an extension other than the one shown will not be visible to the user. In order to see all available files, you will want to have the * symbol in the filter box. When you are finished adding files for analysis, hit **Done**. After the files are selected and you have returned to the **Commands** window, an @ is appended onto the command string. This symbol represents the set of files listed.



2.2 Typing Command Lines

There are two ways to build up commands. You can build commands using buttons and menus. However, this method only provides access to the most basic options, so we only recommend it when you are beginning. Alternatively, you can just type in commands directly to the **Commands** window. Let's try entering a command just by typing. Suppose we want to run an MLU analysis on the sample.cha file. Let us say that we also want to restrict the MLU analysis so that it looks only at the child's utterances. To do this, we enter the following command into the window:

```
mlu +t*CHI sample.cha
```

In this command line, there are three parts. The first part gives the name of the command; the second part tells the program to look at only the *CHI lines; and the third part tells the program which file to analyze as input.

If you press the return key after entering this command, you should see a **CLAN Output** window that gives you the result of this particular MLU analysis. This analysis is conducted, by default, on the %mor line which was generated by the MOR program. If a file does not have this %mor line, then you will have to use other forms of the MLU command that only count utterances in words. Also, you will need to learn how to use the various options, such as +t or +f. One way to learn the options is to use the various buttons in the graphic user interface as a way of learning what CLAN can do. Once you have learned these options, it is often easier to just type in this command directly. However, in other cases, it may be easier to use buttons to locate rare options that are hard to remember. The decision of whether to type directly or to rely on buttons is one that is left to each user.

What if you want to send the output to a permanent file and not just to the temporary **CLAN Output** window? To do this you add the +f switch:

```
mlu +t*CHI +f sample.cha
```

Try entering this command, ending with a carriage return. You should see a message in the **CLAN Output** window telling you that a new file called sample.mlu.cex has been created. If you want to look at that file, type Control-O (Windows) or ⌘-o (Mac) for Open File and you can use the standard navigation window to locate the sample.mlu.cex file. It should be in the same directory as your sample.cha file.

You do not need to worry about the order in which the options appear. In fact, the only order rule that is used for CLAN commands is that the command name must come first. After that, you can put the switches and the file name in any order you wish.

Wildcards

A wildcard uses the asterisk symbol (*) to take the place of something else. For example, if you want to run this command across a group of ten files all ending with the extension .cha, you can enter the command in this form:

```
mlu +tCHI +f *.cha
```

Wildcards can be used to refer to a group of files (*.cha), a group of speakers (CH*), or a group of words with a common form (*ing). To see how these could work together, try out this command:

```
freq *.cha +s”*ing”
```

This command runs the FREQ program on all the .cha files in the LIB directory and looks for all words ending in “-ing.” The output is sent to the **CLAN Output** window and you can set your cursor there and scroll back and forth to see the output. You can print this window or you can save it to a file.

Output Files

When you run the command

```
mlu +f sample.cha
```

the program will create an output file with the name sample.mlu.cex. It drops the .cha extension from the input file and then adds a two-part extension to indicate which command has run (.mlu) and the fact that this is CLAN output file (.cex). If you run this command repeatedly, it will create additional files such as sample.ml0.cex, sample.ml1.cex, sample.ml2.cex, and the like. You can add up to three letters after the +f switch, as in:

```
mlu +fmot sample.cha
```

If you do this, the output file will have the name “sample.mot.cex.” As an example of a case where this would be helpful, consider how you might want to have a group of output files for the speech of the mother and another group for the speech of the father. The mother’s files would be named *.mot.cex and the father’s files would be named *.fat.cex.

Redirection

Instead of using the +f switch for output, you may sometimes want to use the redirect symbol (>). This symbol sends all of the output to a single file. The individual analysis of each file is preserved and grouped into one output file that is named in the command string. The use of the redirect syntax is illustrated in the following examples:

```
freq sample.cha > myanalyses
freq sample.cha >> myanalyses
freq sample.cha >& myanalyses
```

These three forms have slightly different results.

1. The single arrow overwrites material already in the file.
2. The double arrow appends new material to the file, placing it at the end of material already in the file.
3. The single arrow with the ampersand writes both the analyses of the program and various system messages to the file.

If you want to analyze a whole collection of files and send the output from each to a separate file, use the +f switch instead.

2.3 Sample Runs

Now we are ready to try out a few sample runs with the five most basic CLAN commands: FREQ, MLU, COMBO, KWAL, and GEM.

2.3.1 Sample FREQ Runs

FREQ counts the frequencies of words used in selected files. It also calculates the type–token ratio typically used as a measure of lexical diversity. In its simplest mode, it generates an alphabetical list of all the words used by all speakers in a transcript along with the frequency with which these words occur. The following example looks specifically at the child’s tier. The output will be printed in the CLAN window in alphabetical order:

```
freq +t*CHI 0042.cha
```

The output is:

```
> freq +tCHI 0042.cha
freq +tCHI 0042.cha
Tue Aug  7 16:07:59 2007
freq (03-Aug-2007) is conducting analyses on:
  ONLY speaker main tiers matching: *CHI;
****************************************
```

```
From file <0042.cha>
  1 ah
  2 bow+wow
 10 uh
  1 vroom@o
------------------------------
    4  Total number of different word types used
   14  Total number of words (tokens)
0.286  Type/Token ratio
```

A statistical summary is provided at the end. In the above example there were a total of 14 words or tokens used with only five different word types. The type–token ratio is found by dividing the total of unique words by the total of words spoken. For our example, the type–token ratio would be 4 divided by 14 or a ratio of 0.286.

The +f option can be used to save the results to a file. CLAN will automatically add the .frq.cex extension to the new file it creates. By default, FREQ excludes the strings xxx, yyy, www, as well as any string immediately preceded by one of the following symbols: 0, &, +, -, #. However, FREQ includes all retraced material unless otherwise commanded. For example, given this utterance:

```
*CHI: the dog [/] dog barked.
```

FREQ would give a count of two for the word "dog," and one each for the words "the" and "barked." If you wish to exclude retraced material, use the +r6 option. To learn more about the many variations in FREQ, read the section devoted specifically to this useful command.

2.3.2 Sample MLU Run

The MLU command is used primarily to determine the mean length of utterance of a specified speaker. It also provides the total number of utterances and of morphemes in a file. The ratio of morphemes over utterances (MLU) is derived from those two totals. The following command would perform an MLU analysis on the mother's tier (+t*MOT) from the file 0042.cha:

```
mlu +t*MOT 0042.cha
```

The output from this command looks like this:

```
> mlu +tMOT 0042.cha
mlu +tMOT 0042.cha
Tue Aug  7 16:09:52 2007
mlu (03-Aug-2007) is conducting analyses on:
  ONLY speaker main tiers matching: *MOT;
****************************************
From file <0042.cha>
MLU for Speaker: *MOT
   MLU (xxx and yyy are EXCLUDED from the utterance and morpheme
counts):
       Number of: utterances = 509, morphemes = 1406
```

```
        Ratio of morphemes over utterances = 2.762
        Standard deviation = 1.933
```

Thus, we have the mother's MLU or ratio of morphemes over utterances (2.762) and her total number of utterances (509).

2.3.3 Sample COMBO Run

COMBO is a powerful program that searches the data for specified combinations of words or character strings. For example, COMBO will find instances where a speaker says *kitty* twice in a row within a single utterance. The following command would search the mother's tiers (+t*MOT) of the specified file 0042.cha:

```
combo +tMOT +s"kitty^kitty" 0042.cha
```

Here, the string +tMOT selects the mother's speaker tier only for analysis. When searching for a particular combination of words with COMBO, it is necessary to precede the combination with +s (e.g., +s"kitty^kitty") in the command line. The symbol ^ specifies that the word *kitty* is immediately followed by the word *kitty*. The output of the command used above is as follows:

```
> combo +tMOT +s"kitty^kitty" 0042.cha
((kitty^kitty))
combo +tMOT +skitty^kitty 0042.cha
Tue Aug  7 16:11:34 2007
combo (03-Aug-2007) is conducting analyses on:
  ONLY speaker main tiers matching: *MOT;
****************************************
From file <0042.cha>
----------------------------------------
*** File "0042.cha": line 3066.
*MOT:   kitty (1)kitty kitty .
----------------------------------------
*** File "0042.cha": line 3143.
*MOT:   and kitty (1)kitty .

Strings matched 2 times
```

2.3.4 Sample KWAL Run

KWAL searches data for user-specified words and outputs those keywords in context. The +s option is used to specify the words to be searched. The context or cluster is a combination of main tier and the selected dependent tiers in relation to that line. The following command searches for the keyword "bunny" and shows both the two sentences preceding it, and the two sentences following it in the output.

```
kwal +sbunny -w2 +w2 0042.cha
```

The -w and +w options indicate how many lines of text should be included before and after the search words. The output is as follows:

```
> kwal +sbunny -w2 +w2 0042.cha
```

```
Tue Aug  7 16:24:37 2007
kwal (03-Aug-2007) is conducting analyses on:
  ALL speaker tiers
****************************************
From file <0042.cha>
----------------------------------------
*** File "0042.cha": line 2754. Keyword: bunny
*CHI:   0 .
*MOT:   see ?
*MOT:   is the bunny rabbit jumping ?
*MOT:   okay .
*MOT:   wanna [: want to] open the book ?
----------------------------------------
*** File "0042.cha": line 2900. Keyword: bunny
*MOT:   <<one chick breaks out of its shell> ["]> [>] .
*CHI:   <0> [<] .
*MOT:   <and a bunny ges by hoppety+hoppety+hop> ["] .
*MOT:   <<baby koala bear rides on mother's back> ["]> [>] .
*CHI:   <0> [<] .
----------------------------------------
*** File "0042.cha": line 3080. Keyword: bunny
*CHI:   0 .
*MOT:   hmm ?
*MOT:   <the bunny> [>] .
*CHI:   <0> [<] .
*MOT:   <hop hop bunny> [>] .
----------------------------------------
*** File "0042.cha": line 3085. Keyword: bunny
*MOT:   <the bunny> [>] .
*CHI:   <0> [<] .
*MOT:   <hop hop bunny> [>] .
*CHI:   <0> [<] .
*MOT:   you like that book ?
```

2.3.5 Sample GEM Run

GEM searches for previously tagged passages for further analyses. For example, we might want to divide the transcript according to different social situations. By dividing the transcripts in this manner, separate analyses can be conducted on each situation type. One way of doing this is by "piping." Piping directs the output from one command to another.

```
gem +t*CHI +d 0012.cha | freq
```

The output is as follows:

```
> gem +t*CHI +d 0012.cha | freq
Tue Aug  7 16:25:37 2007
gem (03-Aug-2007) is conducting analyses on:
  ONLY speaker main tiers matching: *CHI;
  and ONLY header tiers matching: @BG:; @EG:;
****************************************
From file <0012.cha>
freq
Tue Aug  7 16:25:37 2007
```

```
freq (03-Aug-2007) is conducting analyses on:
  ALL speaker tiers
****************************************
From pipe input
  2 box
  1 byebye
  1 do
  1 going
  1 here
  6 kitty
  2 no+no
  2 oh
  5 this
  1 to
------------------------------
   10  Total number of different word types used
   22  Total number of words (tokens)
0.455  Type/Token ratio
```

The majority of the effort involved in using GEM is in the coding of the gem entries. There are three levels of coding:

1. Lazy GEM is the simplest form of GEM. It needs no @eg because each gem begins with one @g and ends with the next @g.
2. The next level is basic GEM. It can be used when the gem is surrounded by unwanted material. It should be marked with @bg at the beginning and with @eg at the end. Make sure all gems begin with @bg and end with @eg.
3. Tagged gems require the highest degree of care. They are good for identifying speech segments defined by the activities they accompany. They may be embedded with other segments but must be delineated by gem coding with tags to differentiate them from surrounding GEM material.

By using the +t option in the command, you may limit the search to a specific speaker or include the dependent tiers in the output. For example:

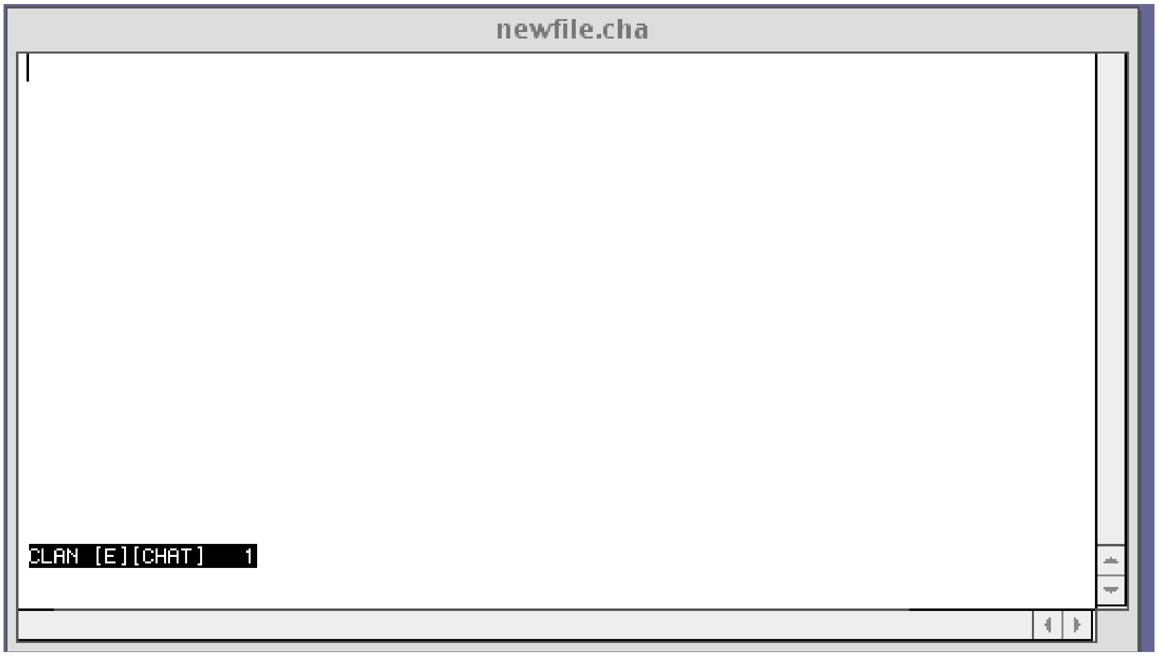
```
gem +t"*MOT" sample.cha
```

The output is as follows:

```
> gem +tMOT sample.cha
Tue Aug  7 16:26:43 2007
gem (03-Aug-2007) is conducting analyses on:
  ONLY speaker main tiers matching: *MOT;
  and ONLY header tiers matching: @BG:; @EG:;
****************************************
From file <sample.cha>
***** From file sample.cha; line 32.
@Bg
*MOT:   what's that ?
*MOT:   is there any delicious cha:lk ?
@Eg
```

3 The Editor

CLAN includes an editor that is specifically designed to work cooperatively with CHAT files. To open up an editor window, either type ⌘-n (Control-n on Windows) for a new file or ⌘-o to open an old file (Control-o on Windows). This is what a new text window looks like on the Macintosh:



You can type into this editor window just as you would in any full-screen text editor.

3.1 Text Mode vs. CHAT Mode

The editor works in two basic modes: Text Mode and CHAT Mode. In Text Mode, the editor functions as a basic ASCII editor. To indicate that you are in Text Mode, the bar at the bottom of the editor window displays [E][Text]. To enter Text Mode, you have to uncheck the CHAT Mode button on the **Mode** pulldown menu. In CHAT Mode, the editor facilitates the typing of new CHAT files and the editing of existing CHAT files. If your file has the extension .cha, you will automatically be placed into CHAT Mode when you open it. To indicate that you are in CHAT Mode, the bar at the bottom of the editor window displays [E][CHAT].

When you are first learning to use the editor, it is best to begin in CHAT mode. When you start CLAN, it automatically opens up a new window for text editing. By default, this file will be opened using CHAT mode. You can use this editor window to start learning the editor or you can open an existing CHAT file using the option in the **File** menu. It is probably easiest to start work with an existing file. To open a file, type *Command-o* (Macintosh) or *Control-o* (Windows). You will be asked to locate a file. Try to open up the sample.cha file that you will find in the Lib directory inside the CLAN directory or folder. This is just a sample file, so you do not need to worry about accidentally saving changes.

You should stay in CHAT mode until you have learned the basic editing commands. You can insert characters by typing in the usual way. Movement of the cursor with the mouse and arrow keys works the same way in this editor as it does in most graphic editors. Functions like scrolling, highlighting, cutting, and pasting also work in the standard way. You should try these functions right away. Use them to move around in the sample.cha file. Try cutting and pasting sections and using the scroll bar, the arrow keys, and the page up and page down keys. Try to type a few sentences.

3.2 File, Edit, and Font Menus

The basic functions of opening files, printing, cutting, undoing, and font changing are common to all window-based text editors. These commands can be found under the **File, Edit,** and **Font** menus in the menu bar. The keyboard shortcuts for pulling down these menu items are listed next to the menu options. Note that there is also a File function called “Save Last Clip As ...” which you can use to save a time-delimited sound segment as a separate file.

3.3 Default Window Positioning and Font Control

Often you may find that you want to control the way in which the CLAN editor displays windows. There are two basic things you may want to control. First, you may wish to control the position of the window on the screen when you first open it. In order to control the default window position, you need to open an empty new file using ⌘-n. Then resize and reposition the window to the form and position that you want and close it. After this, old files that you open will appear in this same position. The system for controlling the default Font is based on a similar idea. Whenever you select a new font, that becomes your current default font. The recent fonts you have used appear at the top of CLAN’s Font menu. If you want to select a new default, just select a new font and that will be the new default.

3.4 CA Styles

CHAT supports many of the CA (Conversation Analysis) codes as developed by Sacks, Schegloff, Jefferson (1974) and their students. The implementation of CA inside CLAN was guided by suggestions from Johannes Wagner, Chris Ramsden, Michael Forrester, Tim Koschmann, Charles Goodwin, and Curt LeBaron. Files that use CA styles should declare this fact by including CA in the @Languages line, as in this example:

```
@Languages:      en, CA
```

It is also useful to apply the CAfont font to CA files. Because the characters in this font have a fixed width, you can use the INDENT program to make sure that CA overlap markers are clearly aligned. Special CA characters can be inserted by typing the F1 function key followed by some letter or number, as indicated in this list:

CA	Char	Function	F1 +	Unicode
up-arrow	↑	shift to high pitch	up arrow	2191
down-arrow	↓	shift to low pitch	down arrow	2193
inverted ?	¿	inhalation	?	00BF
equals sign with slash	≠	no break	=	2260
raised [	⌈	top begin overlap	[	2308
raised]	⌉	top end overlap	]	2309
lowered [	⌊	bottom begin overlap	shift [	230A
lowered]	⌋	bottom end overlap	shift]	230B
up slant arrow	↗	↗ faster ↗	right arrow	2197
down slant arrow	↘	↘ slower ↘	left arrow	2198
single cross	†	† creaky †	t	2020
small super-0	°	° softer °	0 zero	00B0
asterisk	★	★ louder ★	*	2605
low bar	▁	▁ low pitch ▁	d	2581
high bar	▔	▔ high pitch ▔	h	2594
yen sign	\	\ laughed words \	y	00A5
pound sign	£	suppressed laughter pulse	l	00A3
cents sign	¢	pulse of laughter	c	00A2

If you want to convert some of these characters to their more traditional CA forms, you can do this using the CHAT2CA program. However, the files produced by CHAT2CA will no longer be in conformity with CHAT or the CHILDES or TalkBank databases, so we recommend only using this program when you are finished editing the files.

The F1 key is also being used to facilitate insertion of two diacritics for Romanized versions of Arabic. The raised h diacritic is bound to F1-shift-h and the subscript dot is bound to F1-comma.

3.5 Setting Special Colors

Within the Font Menu, you will find options for setting the style of areas as “smaller”, “larger”, “underline”, “italic”, or “color keyword”. It is best to avoid using these formatting features unless necessary, since they tend to complicate the shape of the CHAT file. However, underlining is a crucial component of CA transcription and must be used when you are working in that format. You may also find it important to set the color of certain tiers to improve the readability of your files. For the Macintosh, you can do this in the following way. Select the “Color Keywords” option. In the dialog that appears, type the tier that you want to color in the upper box. For example, you may want to have %mor or *CHI in a special color. Then click on “add to list” and edit the color to the type you wish. The easiest way to do this is to use the crayon selector. Then make sure you select “color entire tier.” To learn the various uses of this dialog, try selecting and applying different options.

3.6 Searching

In the middle of the **Edit** menu, you will find a series of commands for searching. The **Find** command brings up a dialog that allows you to enter a search string and to perform a reverse search. The **Find Same** command allows you to repeat the find multiple times. The **Go To** command allows you to move to a particular line number. The **Replace** command allows you to find a particular string and replace it. There is a dialog on both Macintosh and Windows that allows you to enter your search string, your replacement string, along with tabs or returns. When you need to perform a large series of different replacements, you can set up a file of replacement forms in the two-column form used by CHSTRING. You then are led through the words in this replacement file one by one. On the Macintosh, you have to use the following keyboard commands that are described at the bottom of the editor screen:

!	replace all of them
n	do not replace current occurrence
spacebar	replace the current occurrence
Control-g	abort this command

3.7 Keyboard Commands

In addition to the mouse and the arrow keys, there are many keyboard movement commands based on the EMACS editor. However, most users will prefer to use mouse movements and the commands available in the menu bar. For those familiar with EMACS, a list of these commands can be written out by typing *Esc-h*. This creates a file called keys list that you can then read, save, or print out. If you want to change the binding of a key, you go through these steps:

1. Type *Esc-k*.
2. Enter a command name, such as “cursor-down.”
3. Enter a key, such as F4.
4. Then F4 should move the cursor down.

3.8 Exclude Tiers

This function allows you to hide certain tiers in your transcript. It is equivalent to typing escape-4. If you want to exclude the %mor tier, you type Control-x Control-t (hold down the control key and type x and then t). Then you type e to exclude a tier and %mor for the morphological tier. If you want to exclude all tiers, you type just %. To reset the tiers and to see them all, you type Esc-4 and then r.

3.9 Send to Sound Analyzer.

This mode allows you to send a bulleted sound segment to Praat or Pitchworks. If you are using Praat, you must first start up the Praat window (download Praat from http://www.fon.hum.uva.nl/praat) and place your cursor in front of a bullet for a sound segment. Selecting “send to Praat” then sends that clip to the Praat window for further analysis. To run Praat in the background without a GUI, you can also send this command from a Perl or Tcl script:

```
system ("\"C:\\Program Files\\Praatcon.exe\" myPraatScript.txt
```

3.10 Tiers Menu

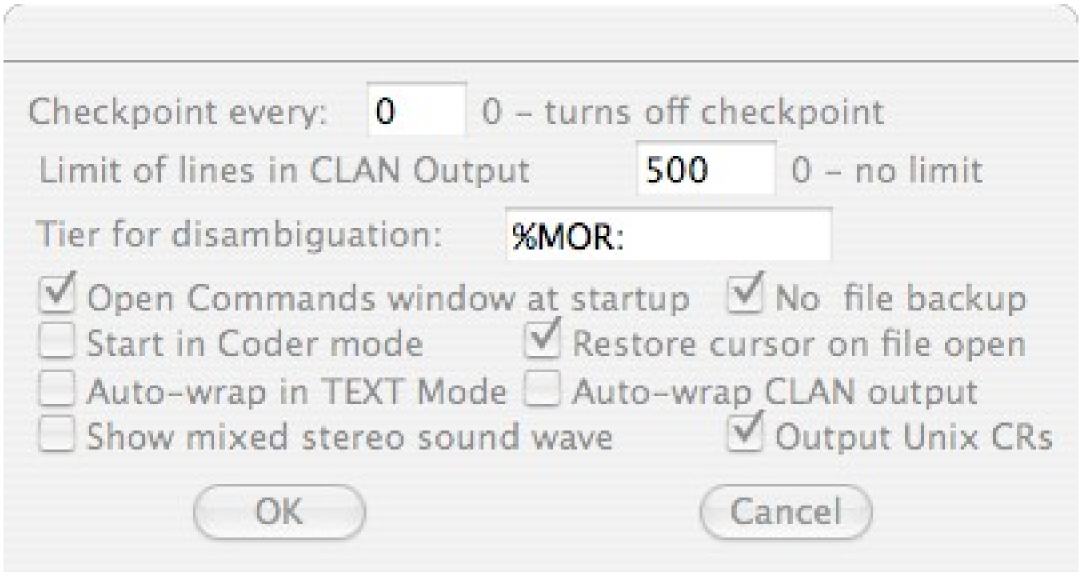
When you open a CHAT file with an @Participants line, the editor looks at each of the participants declared for the file and inserts their codes into the Tiers menu. Each speaker is associated with a keyboard command that lets you enter the name quickly. If you make changes to the @Participants line, you can press the Update button at the bottom of the menu to reload new speaker names.

3.11 Running CHECK Inside the Editor

You can run CHECK from inside the editor. You do this by typing *Esc-L* or selecting **Check Opened File** from the **Mode** menu. If you are in CHAT Mode, CHECK will look for the correct use of CHAT. Make sure that you have set you "Lib" directory to childes/clan/lib, where the depfile.cut file is located. If you are in CA Mode, CHECK will look for the correct use of CA transcription.

3.12 Preferences and Options

You can set preferences by pulling down the **Edit** menu and selecting **Options**. The following dialog box will pop up:



Checkpoint every: 0 0 – turns off checkpoint

Limit of lines in CLAN Output 500 0 – no limit

Tier for disambiguation: %MOR:

☑ Open Commands window at startup ☑ No file backup

☐ Start in Coder mode ☑ Restore cursor on file open

☐ Auto-wrap in TEXT Mode ☐ Auto-wrap CLAN output

☐ Show mixed stereo sound wave ☑ Output Unix CRs

These options control the following features:

1. Checkpoint frequency. This controls how often your file will be saved. If you set the frequency to 50, it will save after each group of 50 characters that you enter.

2. Limit of lines in CLAN output. This determines how many output lines will go to your CLAN output screen. It is good to use a large number, since this will allow you to scroll backwards through large output results.
3. Tier for disambiguation. This is the default tier for the Disambiguator Mode function.
4. Open Commands window at startup. Selecting this option makes it so that the **Commands** window comes up automatically whenever you open CLAN.
5. No backup file. By default, the editor creates a backup file, in case the program hangs. If you check this, CLAN will not create a backup file.
6. Start in CHAT Coder mode. Checking this will start you in Text Mode when you open a new text window.
7. Auto-wrap in Text Mode. This will wrap long lines when you type.
8. Auto-wrap CLAN output. This will wrap long lines in the output.
9. Show mixed stereo sound wave. CLAN can only display a single sound wave when editing. If you are using a stereo sound, you may want to choose this option.
10. Output Unix CRs. This is for people who use CLAN on Unix.

4 Linkage

In the old days, transcribers would use a foot pedal to control the rewinding and replaying of tapes. With the advent of digitized audio and video, it is now possible to use the computer to control the replay of sound during transcription. Moreover, it is possible to link specific segments of the digitized audio or video to segments of the computerized transcript. This linkage is achieved by inserting a header tier of this shape

```
@Media: clip, audio
```

The first field in the @Media line is the name of the media file. You do not need to include the extension of the media file name. Each transcript should be associated with one and only one media file. To keep your project well organized it is best if the media file name matches the transcript file name. The second field in the @Media header tells whether the media is audio, video, or missing.

Once this header tier is entered, you can use various methods to insert sound markers that appear initially to the user as bullets. When these bullets are opened up they look like this:

```
*ROS:   alert [!] alert !  ·1927_4086·
```

When then are closed then look like this:

```
*ROS:   alert [!] alert ! ·
```

The size and shape of the bullet character varies across different fonts, but it will usually be a bit darker than what you see above. The information in the bullet provides clearer transcription and immediate playback directly from the transcript. The first number in the bullet indicates the beginning of the segment in milliseconds and the second number indicates the end in milliseconds.

Once a CHAT files has been linked to audio or video, it is easy to playback the interaction from the transcript using “Continuous Playback” mode (escape-8). In this mode, the waveform display is turned off and the computer plays back the entire transcript, one utterance after another, while moving the cursor and adjusting the screen to continually display the current utterances. This has the effect of “following the bouncing ball” as in the old sing-along cartoons or karaoke video. In Continuous Movie Playback Mode, the video is played as the cursor highlights utterances in the text.

To create a text that can be played back and studied in this way, however, the user can make use of any combination of six separate methods: sonic mode, transcriber mode, video mode, sound walker, time mark editing, and export to partitur editors. This chapter describes each of these five methods and leaves it up to the individual researcher which of these methods is best for his or her project.

To use any of these methods, you need to have a digitized audio or video file. Audio files can be in either .wav or .mp3 format. Video files can be in any video format that can be played by QuickTime. You will also need to have QuickTime installed on your machine. Once you have created a digitized sound file for the material you wish to transcribe, you are ready to start using one of the five methods described below.

4.1 Sonic Mode

Sonic Mode involves transcribing from a sound waveform. Currently, Sonic Mode can only be used with audio files. To begin Sonic transcription, you should launch CLAN and open a new file. Type in your basic header tiers first, along with the @Media header discussed above. Then, go to the **Mode** pulldown menu and select "Sonic Mode" and you will be asked to locate the digitized sound file. Once you have selected your file, the waveform comes up, starting at the beginning of the file. Several functions are available at this point:

1. **Sound playing from the waveform.** You can drag your cursor over a segment of the waveform to highlight it. When you release your mouse, the segment will play. As long as it stays highlighted, you can replay it by holding down the shift key and clicking the mouse. At this point, it does not matter where your cursor is positioned.
2. **Waveform demarcation.** You can move the borders of a highlighted region by holding down the shift key and clicking your mouse to place the cursor at the place to which you wish the region to move. You can use this method to either expand or contract the highlighted region.
3. **Transcription.** While you are working with the waveform, you can repeatedly play the sound by using shift-click. This will help you recognize the utterance you are trying to transcribe. You then go back to the editor window and type out the utterance that corresponds to the highlighted segment.
4. **Linking.** When you believe that the highlighted waveform corresponds correctly to the utterance you have transcribed, you can click on the "s" button to the left of the waveform display and a bullet will be inserted. This bullet contains information regarding the exact onset and offset of the highlighted segment. You can achieve the same effect using escape-I (insert time code).
5. **Changing the waveform window.** The **+H** and **-H** buttons on the left allow you to increase or decrease the amount of time displayed in the window. The **+V** and **-V** buttons allow you to control the amplitude of the waveform.
6. **Scrolling.** At the bottom of the sound window is a scroll-bar that allows you to move forward or backward in the sound file (please note that scrolling in the sound file can take some time as the sound files for long recordings are very large and take up processing capacity).
7. **Waveform activation**. In order to highlight the section of the waveform associated with a particular utterance, you need to triple-click on the bullet following the utterance you want to replay. You must triple-click at a point just before the bullet to get reliable movement of the waveform. If you do this correctly, the waveform will redisplay. Then you can replay it by using shift-click.
8. **Expanding and hiding the bullets**. If you want to see the exact temporal references that are hiding inside the bullet symbols, you can type *Esc-A* to expand them. Typing *Esc-A* again will hide them again.
9. **Time duration information**. Just above the waveform, you will see the editor

mode line. This is the black line that begins with the word “CLAN”. If you click on this line, you will see three additional numbers. The first is the beginning and end time of the current window in seconds. The second is the position of the cursor in hours:minutes:seconds.milliseconds. The third is the beginning and end of the current selection in seconds. If you click once again on the mode line, you will see sampling rate information for the audio file.

4.2 Transcriber Mode

This mode is faster than Sonic or Video Mode, but often less precise. However, unlike Sonic Mode, it can also be used for video transcription. Transcriber Mode is intended for two uses. The first is for transcribers who wish to link a digitized file to an already existing CHAT transcript. The second is for transcribers who wish to produce a new transcript from a digitized file.

4.2.1 Linking to an already existing transcript

To link a video or audio file to an already existing transcript, please follow these steps:

1. Place your CLAN transcript and video file to the same directory.
2. Set your working directory as the location of the video and transcript.
3. Open the CLAN transcript.
4. Enter a few basic headers, including the @Media header discussed at the beginning of this chapter.
5. Place your cursor somewhere within the first utterance.
6. Click on Mode, Transcribe Sound or Movie or just type F5.
7. When CLAN asks you for the movie, click on the video file you want to transcribe.
8. The movie will automatically start playing in a Quicktime video player. When it does, listen for the different utterances. At the end of each utterance, press the spacebar. This will automatically record a bullet at the end of the line that “connects” the video to the transcript.
9. If you get off at any point in time, click on the video window and the video will stop running.
10. Once playback is stopped, reposition your cursor at the last correct bullet and again click on “Transcribe sound or movie.” The movie will automatically begin at the bullet where you cursor is. As you press the spacebar, the new bullets will overwrite the old ones.
11. After you have spent some time inserting bullets, click file, save. The bullets will be saved into your transcript.
12. After you are done adding bullets, click in the video window to stop the process. Then go to the top of the file, and insert @Begin and @Participants lines. Use the @Participants to generate key shortcuts under the View menu. Then replay the first bullet, transcribe it, and use the appropriate command-1 or command-2 key to enter the speaker ID. Then go on to the next utterance and repeat the process. The result will be a full transcription that is roughly linked to the audio.

4.2.2 To create a new transcript

You can use the F5 insertion mode to create a new transcript in which the links have already been inserted. Here are the steps:

1. Open a blank file.
2. Enter a few basic headers, along with the @Media header discussed at the beginning of this chapter. Make sure that the media is in your working directory.
3. Go to "Mode," and select "Transcribe Sound or Movie {F5}."
4. Find the movie clip you want to transcribe.
5. When you click on it, the movie will open in another window.
6. Immediately, start pressing the spacebar at the end of each utterance. This will insert a bullet into the blank transcript.
7. When you are finished inserting the bullets, save the file.
8. Then, one at a time, Crtl-click on the bullets and transcribe them.
9. If the bullets need some adjusting, you may do this while you are transcribing by manipulating the numbers in the movie window and clicking on the save button in the lower right hand corner. You can also expand the bullets {Esc-A} and type it in manually.
10. Save frequently.
11. When you are done transcribing, it is a good idea to look at the transcript in continuous playback mode to make sure everything is transcribed correctly. To do this go to "Mode," and then "Continuous Playback {Esc-8}."

4.2.3 Sparse Annotation

For some applications, it is not necessary to produce a complete transcription of an interaction. Instead, it is sufficient to link a few comments to just a few important segments. For example, during a one-hour classroom Mathematics lesson, it might be sufficient to point out just a few "teachable moments." To do this, you can follow these steps:

1. open a new file.
2. insert a few basic headers, along with the @Media header discussed at the beginning of this chapter. Make sure the media is in your working directory.
3. select the "Edit" menu and pulldown to "Select F5 option"
4. in segment length type 3000 for 3 seconds bullet length press OK button
5. start F5 mode by pressing F5 key
6. select the media you want and CLAN will start playing.
7. when you hear what you want to comment click F1 or F2 or F5
8. the bullet will be inserted into text and playback will stop
9. click on transcript text window and add comment to that bullet.
10. move text cursor to the last bullet less tier, i.e. "*: ".
11. press F5 again and the process will start again from last stop.

4.3 Video Linking

To learn to do video linking, it is best to first the video.zip file that contains transcripts and a QuickTime movie for the first 28 utterances in the "MyTheory" problem-based learning session analyzed in a special issue of Discourse Processes (27:2) edited by Tim Koschmann. This file is at http://www.talkbank.org/dv/final.html. Using video.zip, you then:

1. Unzip the file.
2. If you have a recent version of CLAN and QuickTime 7 or later, just double click on 28lines.cha. If you don't have QuickTime 7 or above, you can download a copy from http://www.apple.com/quicktime/download/
3. After the file opens, you will see some solid black bullets. These are the time markers. If you type escape-A they will expand and you can see their values. Then type escape-A again to close them.
4. Now place your cursor near the beginning of the file and type escape-8 for continuous playback. The video window should open and the file should play back. When you want it to stop, double click.
5. To play back a single line, place your mouse just to the left of a bullet, press the command key (Windows: control) and click the mouse. After a few seconds of pause, the segment will play.
6. Try playing different segments, as well as using the continuous movie playback mode, showing and playing from thumbnails, and other video features found in the "Mode" menu.

If you want to link your transcript to a movie or create a new transcript that is linked to a movie, you can use one of two methods –Transcriber Mode or Manual Linking Mode. Transcriber Mode was described in the previous section. It is a quick and easy method that will prove useful for beginning linking to a particular transcript. Using this method, however, sacrifices precision. It is then necessary to go back and tighten up the links using the Manual Linking method. The Help screen on the video window gives you the functions you will need for this. Many of these functions apply to both video and audio. Their use is summarized here:

1. <- will set back the current time. This function makes small changes at first and then larger ones if you keep it pressed down.
2. -> will advance the current time. This function makes small changes at first and then larger ones if you keep it pressed down.
3. control <- will decrease the beginning value for the segment in the text window as well as the beginning value for the media in the video window. This function makes small changes at first and then larger ones if you keep it pressed down.
4. control -> will increase the beginning value for the segment in the text window as well as the beginning value for the media in the video window. This function makes small changes at first and then larger ones if you keep it pressed down.
5. command <- will decrease the beginning value for the segment in the text window as well as the beginning value for the media in the video window. This function makes small changes at first and then larger ones if you keep it pressed down.

6. command -> will increase the beginning value for the segment in the text window as well as the beginning value for the media in the video window. This function makes small changes at first and then larger ones if you keep it pressed down.
7. / pressing the button with the right slash with the start time active moves the start time to current time. If the current time is active, it moves the current time to the start time.
8. \ pressing the button with the left slash with the end time active moves the end time to current time. If the current time is active, it moves the current time to the end time.
9. Triple-clicking on the relevant cell has the same effect as the above two functions.
    10. You can play the current segment either by pressing the repeat button or the space button when the video window is active. The behavior of the repeat play function can be altered by inserting various values in the box to the right of "repeat". These are illustrated in this way:

-400	add 400 milliseconds to the beginning of the segment to be repeated
+400	add 400 milliseconds to the end of the segment to be repeated
b400	play the first 400 milliseconds of the segment
e400	play the last 400 milliseconds of the segment

4.4 SoundWalker

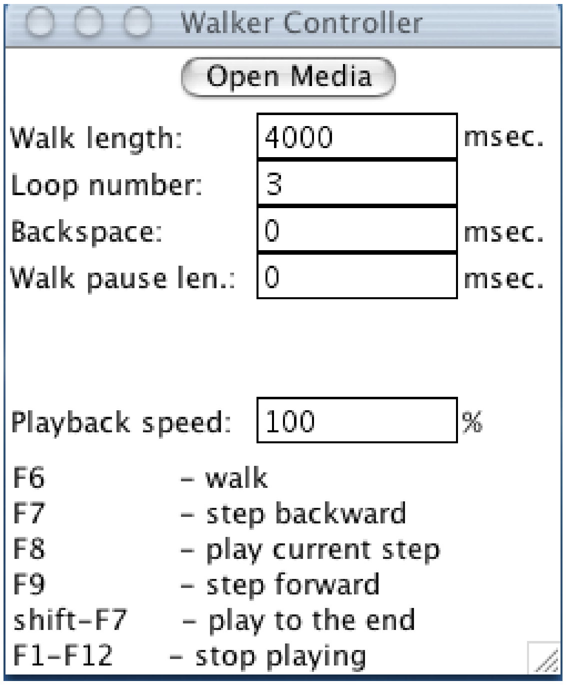
The SoundWalker facility is based on the conception implemented by Jack DuBois at UC Santa Barbara. This controller allows you to step forward and backwards through a digitized file, using a few function keys. It attempts to imitate the old transcriber foot pedal, but with some additional functionality. The options you can set are:

1. walk length: This sets how long a segment you want to have repeated.
2. loop number: If you set 3, for example, the programs plays each step three times before moving on.
3. backspace: The amount of rewinding in milliseconds at the end of each loop.
4. walk pause length: The duration of the pause between loops.
5. playback speed: This setting allows you to speed up or slow down your playback rate.

The basic contrast here is between "stepping" which means moving one step forward or back and "walking" which just keeps on stepping one step after another in the manner you have specified with the above option. The keys you use are:

F6	walk
F7	step backward
F8	play current step
F9	step forward
shift F7	play to the end
F1-F12	stop playing

You will find all of these options in the "Walker Controller" dialog that you open under the Window menu. Once you open the Walker Controller and thereby enable SoundWalker, the functions described above become enabled.



Walk length: 4000 msec.
Loop number: 3
Backspace: 0 msec.
Walk pause len.: 0 msec.
Playback speed: 100 %

F6 - walk
F7 - step backward
F8 - play current step
F9 - step forward
shift-F7 - play to the end
F1-F12 - stop playing

If you would like to use a real foot pedal with SoundWalker, you can order one (Windows Only) from www.xkeys.com. This foot pedal installs along with the keyboard and allows you to bind F6, F7, and F8 to the left, middle, and right pedals for the functions of rewind, play, and forward.

4.5 *Export to Partitur Editors*

Although CLAN is an excellent tool for basic transcription and linkage, more fine-grained control for overlap marking is best achieved in a Partitur or “musical score” editor like EXMARaLDA or ELAN. EXMARaLDA can be downloaded from www1.uni-hamburg.de/exmaralda/and ELAN can be downloaded from www.mpi.nl/tools/. To convert CHAT files to and from ELAN and EXMARaLDA, you can use the CHAT2ELAN, ELAN2CHAT, CHAT2EXMAR, and EXMAR2CHAT programs. When editing in ELAN and EXMARaLDA, it is best to focus only on the use of these tools for time alignment and not for other aspects of editing, since CLAN is more highly structured for these other aspects of transcription.

4.6 *Playback Control*

Once a transcript has been linked, you will want to study it through playback. Basic playback uses the escape-8 command for continuous playback and command-click for playback of a single utterance. You can use manual editing of the bullets to modify playback in two other ways:

4.6.1 Forced Skipping

If you have an audio recording with a large segment that you do not wish to transcribe, you may also wish to exempt this segment from continuous playback. If you do not apply forced skipping to do this, you will have to listen through this untranscribed material during continuous playback. To implement forced skipping, you should open up your bullets using escape-A. Then go to the end of the bullet after the skip and insert a dash after the second time value. For example, you might change 4035_5230 to 4035_5230 Then close the bullets and save the file.

5 Coder Mode

Coder Mode is useful for researchers who have defined a fully structured coding scheme that they wish to apply to all the utterances in a transcript. To begin Coder Mode, you need to shift out of Editor Mode. To verify your current mode, just double-click on a file. Near the bottom of the text window is a line like this:

```
CLAN [E] [chat] sample.cha 1
```

The [E] entry indicates that you are in editor mode and the [chat] entry indicates that you are in CHAT Mode. In order to begin coding, you first want to set your cursor on the first utterance you want to code. You can use an file to do this. If the file already has %spa lines coded, you will be adding additional codes. If none are present yet, Coder's Editor will be adding new %spa line. Once you have placed the cursor anywhere on the first line you want to code, you are ready to leave CHAT Mode and start using Coder Mode. To go into Coder Mode, type Esc-e. You will be asked to load a codes file. Just navigate to your library directory and select one of the demo codes files beginning with the word "code." We will use codes1.cut for our example.

5.1.1 Entering Codes

Now the coding tier that appears at the top line of the codes1.cut file is shown at the bottom of the screen. In this case it is %spa:. You can either double-click this symbol or just hit the carriage return and the editor will insert the appropriate coding tier header (e.g. %spa), a colon and a tab on the line following the main line. Next it will display the codes at the top level of your coding scheme. In this case, they are $POS and $NEG. You can select one of these codes by using either the cursor keys, the plus and minus keys or a mouse click. If a code is selected, it will be highlighted. You can enter it by hitting the carriage return or double-clicking it. Next, we see the second level of the coding scheme.

To get a quick overview of your coding choices, type *Esc-s* several times in succession and you will see the various levels of your coding hierarchy. Then return back to the top level to make your first selection. When you are ready to select a top-level code, double-click on it with your mouse. Once you have selected a code on the top level of the hierarchy, the coder moves down to the next level and you repeat the process until that complete code is constructed. To test this out, try to construct the code $POS:COM:VE.

The coding scheme entered in codes1.cut is hierarchical, and you are expected to go through all the decisions in the hierarchy. However, if you do not wish to code lower levels, type *Esc-c* to signal that you have completed the current code. You may then enter any subsequent codes for the current tier.

Once you have entered all the codes for a particular tier, type *Esc-c* to signal that you are finished coding the current tier. You may then either highlight a different coding tier relevant to the same main line, or move on to code another main line. To move on to another main line, you may use the arrow keys to move the cursor or you may automatically proceed to next main speaker tier by typing *Control-t*. Typing *Control-t* will move the cursor to the next main line, insert the highlighted dependent coding tier, and position you to select a code from the list of codes given. If you want to move to yet another line, skipping over a line, type *Control-t* again. Try out these various commands to see how they work.

If you want to code data for only one speaker, you can restrict the way in which the *Control-t* feature works by using *Esc-t* to reset the set-next-tier-name function. For example, you confine the operation of the coder to only the *CHI lines, by typing *Esc-t* and then entering *CHI*. You can only do this when you are ready to move on to the next line.

If you receive the message "Finish coding current tier" in response to a command (as, for example, when trying to change to editor mode), use *Esc-c* to extricate yourself from the coding process. At that point, you can reissue your original command. Here is a summary of the commands for controlling the coding window.

Function
finish coding current code
finish coding current tier
finish coding current tier and go to the next
restrict coding to a particular speaker
go on to the next speaker
show subcodes under cursor

5.1.2 Setting Up Your Codes File

When you are ready to begin serious coding, you will want to create your own codes file to replace our sample. When editing your codes file, make sure that you are in Text Mode and not CHAT Mode. You select Text Mode from the menu by deselecting (unchecking) CHAT Mode in the Mode menu. To make sure you are in Text Mode, look for [E][TEXT] in the bottom line of the Editor window. If you decide to use another editor or if you do not use Text Mode in CLAN, you will probably have problems.

You will probably find it useful to use the samples in the /coder folder in the /lib folder included with the CLAN program distribution. In the next paragraphs, we will explain the construction of the codes-basic.cut file in that folder. The first line of your codes-basic.cut file is:

```
\ +b50 +d +l1 +s1
```

The options on the main line were described in the previous section on editor options. In this example, the +b option sets the checkpoint buffer (that is, the interval at which the program will automatically back up the work you have done so far in that session). If you find the interval is too long or too short, you can adjust it by changing the value of b. The +d option tells the editor to keep a ".bak" backup of your original CHAT file. To turn off the backup option, use –d. The +l option reorders the presentation of the codes based on their frequency of occurrence. There are three values of the +l option:

0 leave codes without frequency ordering
1 move most frequent code to the top
2 move codes up one level by frequency

If you use the +s option, the program assumes that all of the codes at a particular level have the same codes symmetrically nested within them. For example, consider the codes-basic.cut file:

```
\ +l1 +s1 +b50
%spa:
 " $MOT
  :POS
   :Que
   :Res
  :NEG
 " $CHI
```

The spaces in this file must be spaces and not tabs. However, there must be a tab following the colon on the %spa: tier. The above file is a shorthand for the following complete listing of code types:

```
$MOT:POS:Que
$MOT:POS:Res
$MOT:NEG:Que
$MOT:NEG:Res
$CHI:POS:Que
$CHI:POS:Res
$CHI:NEG:Que
$CHI:NEG:Res
```

It is not necessary to explicitly type out each of the eight combinations of codes. With the +s1 switch turned on, each code at a particular level is copied across the branches so that all of the siblings on a given level have the same set of offspring. A more extensive example of a file that uses this type of inheritance is the system for error coding given in the codeserr.cut file in the /lib/coder folder distributed with CLAN.

If not all codes at a given level occur within each of the codes at the next highest level, each individual combination must be spelled out explicitly and the +s option should not be used. The second line in the file should declare the name for your dependent tier. It should end with a tab, so that the tab is inserted automatically in the line you are constructing. A single codes.cut file can include coding systems for many different dependent tiers with each system in order in the file and beginning with an identifier such as $spa:.

Setting up the codes.cut file properly is the trickiest part of Coder Mode. Once properly specified, however, it rarely requires modification. If you have problems getting the editor to work, chances are the problem is with your codes.cut file.

6 Exercises

This chapter presents exercises designed to help you think about the application of CLAN for specific aspects of language analysis. The illustrations in the section below are based on materials developed by Barbara Pan originally published in Chapter 2 of Sokolov and Snow (1994). The original text has been edited to reflect subsequent changes in the programs and the database. Barbara Pan devised the initial form of this extremely useful set of exercises and kindly consented to their inclusion here.

6.1 Contrasting Four Measures

One approach to transcript analysis focuses on the computation of particular measures or scores that characterize the stage of language development in the children or adults in the sample.

1. One popular measure (Brown, 1973) is the MLU or mean length of utterance, which can be computed by the MLU program.
2. A second measure is the MLU of the five longest utterances in a sample, or MLU5. Wells (1981) found that increases in MLU of the five longest utterances tend to parallel those in MLU, with both levelling off after about 42 months of age. Brown suggested that MLU of the longest utterance tends, in children developing normally, to be approximately three times greater than MLU.
3. A third measure is MLT or Mean Length of Turn which can be computed the the MLT program.
4. A fourth popular measure of lexical diversity is the type–token ratio of Templin (1957).

In these exercises, we will use CLAN to generate these four measures of spontaneous language production for a group of normally developing children at 20 months. The goals are to use data from a sizeable sample of normally developing children to inform us as to the average (mean) performance and degree of variation (standard deviation) among children at this age on each measure; and to explore whether individual children's performance relative to their peers was constant across domains. That is, were children whose MLU was low relative to their peers also low in terms of lexical diversity and conversational participation? Conversely, were children with relatively advanced syntactic skills as measured by MLU also relatively advanced in terms of lexical diversity and the share of the conversational load they assumed?

The speech samples analyzed here are taken from the New England corpus of the CHILDES database, which includes longitudinal data on 52 normally-developing children. Spontaneous speech of the children interacting with their mothers was collected in a play setting when the children were 14, 20, and 32 months of age. Transcripts were prepared according to the CHAT conventions of the Child Language Data Exchange System, including conventions for morphemicizing speech, such that MLU could be computed in terms of morphemes rather than words. Data were available for 48 of the 52 children at 20 months. The means and standard deviations for MLU5, TTR, and MLT reported below are based on these 48 children. Because only 33 of the 48 children produced 50 or more utterances during the observation session at 20 months, the mean and standard deviation for MLU50 is based on 33 subjects.

For illustrative purposes, we will discuss five children: the child whose MLU was the highest for the group (68.cha), the child whose MLU was the lowest (98.cha), and one child each at the first (66.cha), second (55.cha), and third (14.cha) quartiles. Transcripts for these five children at 20 months can be found in the /ne20 directory in the /lib directory distributed with CLAN.

Our goal is to compile the following basic measures for each of the five target children: MLU on 50 utterances, MLU of the five longest utterances, TTR, and MLT. We then compare these five children to their peers by generating *z*-scores based on the means and standard deviations for the available sample for each measure at 20 months. In this way, we were will generate language profiles for each of our five target children.

6.2 MLU50 Analysis

The first CLAN analysis we will perform calculates MLU for each child on a sample of 50 utterances. By default, the MLU program excludes the strings xxx, yyy, www, as well as any string immediately preceded by one of the following symbols: 0, &, +, -, #, $, or : (see the CHAT manual for a description of transcription conventions). The MLU program also excludes from all counts material in angle brackets followed by [/], [//], or [% bch] (see the CLAN manual for list of symbols CLAN considers to be word, morpheme, or utterance delimiters). Remember that to perform any CLAN analysis, you need to be in the directory where your data is when you issue the appropriate CLAN command. In this case, we want to be in /childes/clan/lib/ne20. The command string we used to compute MLU for all five children is:

```
mlu +t*CHI +z50u +f *.cha
+t*CHI          Analyze the child speaker tier only
+z50u           Analyze the first 50 utterances only
+f              Save the results in a file
*.cha           Analyze all files ending with the extension .cha
```

The only constraint on the order of elements in a CLAN command is that the name of the program (here, MLU) must come first. Many users find it good practice to put the name of the file on which the analysis is to be performed last, so that they can tell at a glance both what program was used and what file(s) were analyzed. Other elements may come in any order.

The option +t*CHI tells CLAN that we want only CHI speaker tiers considered in the analysis. Were we to omit this string, a composite MLU would be computed for all speakers in the file.

The option + z50u tells CLAN to compute MLU on only the first 50 utterances. We could, of course, have specified the child's first 100 utterances (+z100u) or utterances from the 51st through the 100th (+z51u-100u). With no +z option specified, MLU is computed on the entire file.

The option +f tells CLAN that we want the output recorded in output files, rather than simply displayed onscreen. CLAN will create a separate output file for each file on which it computes MLU. If we wish, we may specify a three-letter file extension for the output files immediately following the +f option in the command line. If a specific file extension is not specified, CLAN will assign one automatically. In the case of MLU, the default extension is .mlu.cex. The .cex at the end is mostly important for Windows, since it allows the Windows operating system to know that this is a CLAN output file.

Finally, the string *.cha tells CLAN to perform the analysis specified on each file ending in the extension .cha found in the current directory. To perform the analysis on a single file, we would specify the entire file name (e.g., 68.cha). It was possible to use the wildcard * in this and following analyses, rather than specifying each file separately, because:

1. All the files to be analyzed ended with the same file extensions and were in the same directory; and
2. in each file, the target child was identified by the same speaker code (i.e., CHI), thus allowing us to specify the child's tier by means of +t*CHI.

Utilization of wildcards whenever possible is more efficient than repeatedly typing in similar commands. It also cuts down on typing errors.

By default, CLAN computes MLU in morphemes, rather than words, if the transcript is morphemicized on the main line. The user may override this default and have CLAN ignore morphemicization symbols by using the option, followed by those symbols to be ignored. For example, -c# would instruct CLAN to ignore the prefix symbol in words such as un#tie; -c#-would result in both the # and - symbols in un#tie-ed being disregarded. Thus, researchers can choose not to count morphemes they believe the child is not yet using productively. To have all morphemicization symbols ignored, one would use -c#&- .

For illustrative purposes, let us suppose that we ran the above analysis on only a single child (68.cha), rather than for all five children at once (by specifying *.cha). We would use the following command:

```
mlu +t*CHI +z50u 68.cha
```

The output for this command would be as follows:

```
> mlu +t*CHI +z50u 68.cha
mlu +t*CHI +z50U 68.cha
Wed Oct 20 11:46:51 1999
mlu (18-OCT-99) is conducting analyses on:
  ONLY speaker main tiers matching: *CHI;
****************************************
From file <68.cha>
```

```
MLU for Speaker: *CHI:
   MLU (xxx and yyy are EXCLUDED from the utterance and morpheme
counts):
        Number of: utterances = 50, morphemes = 133
        Ratio of morphemes over utterances = 2.660
        Standard deviation = 1.570
```

MLU reports the number of utterances (in this case, the 50 utterances we specified), the number of morphemes that occurred in those 50 utterances, the ratio of morphemes over utterances (MLU in morphemes), and the standard deviation of utterance length in morphemes. The standard deviation statistic gives some indication of how variable the child's utterance length is. This child's average utterance is 2.660 morphemes long, with a standard deviation of 1.570 morphemes.

Check line 1 of the output for typing errors in entering the command string. Check lines 3 and possibly 4 of the output to be sure the proper speaker tier and input file(s) were specified. Also check to be sure that the number of utterances or words reported is what was specified in the command line. If CLAN finds that the transcript contains fewer utterances or words than the number specified with the +z option, it will still run the analysis but will report the actual number of utterances or words analyzed.

6.3 MLU5 Analysis

The second CLAN analysis we will perform computes the mean length in morphemes of each child's five longest utterances. To do this, we direct the output of one program to a second program for further analysis. This process is called piping. Although we could accomplish the same goal by running the first program on each file, sending the output to files and then performing the second analysis on the output files, piping is more efficient. The trade-off is that the analysis must be done on one file at a time (by specifying the full file name), rather than by using the * wildcard. The CLAN command string we use is:

```
maxwd +t*CHI +g1 +c5 +dl 68.cha | mlu > 68.ml5.cex
```

+t*CHI	Analyze the child speaker tier only
+gl	Identify the longest utterances in terms of morphemes
+c5	Identify the five longest utterances
+dl	Output the data in CHAT format
68.cha	The child language transcript to be analyzed
| mlu	Pipe the output to the MLU program
>	Send the output of MLU to a file
68.ml5.cex	Create a file for the output, called ml5.cex

If we run simply the first part of this command up to the pipe symbol, the output would look like this:

```
*CHI:   <I want to see the other box> [?] .
*CHI:   that-'is [= book] the <morning # noon and night> ["] .
*CHI:   there-'is a dolly in there [= box] .
*CHI:   it-'is [= contents of box] crayon-s and paper .
```

```
*CHI:   pop go-es the weasel .
```

By adding the MLU command after the pipe, we are telling CLAN to take this initial output from MAXWD and send it on for further processing by MLU.

The string +g1 tells MAXWD to identify longest utterances in terms of morphemes per utterance. If length is to be determined instead by the number of words per utterance, the string +g2 would be used; if by number of characters per utterance, +g3 would be used. For the +g1 switch to work well, we need to either break words into morphemes on the main line (as described in the CHAT manual) or else run this command on the %mor line.

The string +c5 tells MAXWD to identify the five longest utterances.

The string +d1 tells MAXWD to send output to the output file in CHAT form, that is, in a form that can be analyzed by other CLAN programs.

The piping symbol | (upright bar or vertical hyphens) separates the first CLAN command from the second, and indicates that the output of the first command is to be used as the input to the second.

Finally, the redirect symbol > followed by the output file name and extension specifies where the final output file is to be directed (i.e., saved). Omission of the redirect symbol and file name will result in output being displayed on-screen rather than recorded in a file. Here we are specifying that the output from MLU should be recorded in an output file called 68.ml5.cex. The contents of this file are as follows:

```
MLU for Speaker: *CHI:
   MLU (xxx and yyy are EXCLUDED from the utterance and morpheme
counts):
        Number of: utterances = 5, morphemes = 31
        Ratio of morphemes over utterances = 6.200
        Standard deviation = 0.748
```

The procedure for obtaining output files in CHAT format differs from program to program but it is always the +d option that performs this operation. You must check the +d options for each program to determine the exact level of the +d option that is required. We can create a single file to run this type of analysis. This is called a batch file. The batch file for this particular analysis would be:

```
maxwd +t*CHI +g1 +c5 +dl 14.cha | mlu > 14.ml5.cex
maxwd +t*CHI +g1 +c5 +dl 55.cha | mlu > 55.ml5.cex
maxwd +t*CHI +g1 +c5 +dl 66.cha | mlu > 66.ml5.cex
maxwd +t*CHI +g1 +c5 +dl 68.cha | mlu > 68.ml5.cex
maxwd +t*CHI +g1 +c5 +dl 98.cha | mlu > 98.ml5.cex
```

To run all five commands in sequence automatically, we put the batch file in our working directory with a name such as batchml5.cex and then enter the command

```
batch batchml5
```

This command will produce five output files.

6.4 MLT Analysis

The third analysis we will perform is to compute MLT (Mean Length of Turn) for both child and mother. Note that, unlike the MLU program, the CLAN program MLT includes the symbols xxx and yyy in all counts. Thus, utterances that consist of only unintelligible vocal material still constitute turns, as do nonverbal turns indicated by the postcode [+ trn] as illustrated in the following example:

```
*CHI:   0.[+ trn]
%gpx:   CHI points to picture in book
```

We can use a single command to run our complete analysis and put all the results into a single file.

```
mlt *.cha > allmlt.cex
```

In this output file, the results for the mother in 68.cha are:

```
MLT for Speaker: *MOT:
   MLT (xxx and yyy are INCLUDED in the utterance and morpheme
counts):
 Number of: utterances = 331, turns = 227, words = 1398
       Ratio of words over turns = 6.159
       Ratio of utterances over turns = 1.458
       Ratio of words over utterances = 4.224
```

There is similar output data for the child. This output allows us to consider Mean Length of Turn either in terms of words per turn or utterances per turn. We chose to use words per turn in calculating the ratio of child MLT to mother MLT, reasoning that words per turn is likely to be sensitive for a somewhat longer developmental period. MLT ratio, then, was calculated as the ratio of child MLT over mother MLT. As the child begins to assume a more equal share of the conversational load, the MLT ratio should approach 1.00. For this example, the MLT ratio would be: 2.241 ÷ 6.159 = 0.3638.

6.5 TTR Analysis

The fourth CLAN analysis we will perform for each child is to compute the TTR or-type–token ratio. For this we will use the FREQ command. By default, FREQ ignores the strings xxx (unintelligible speech) and www (irrelevant speech researcher chose not to transcribe). It also ignores words beginning with the symbols 0, &, +, -, or #. Here we were interested not in whether the child uses plurals or past tenses, but how many different vocabulary items she uses. Therefore, we wanted to count “cats” and “cat” as two tokens (i.e., instances) of the word-type “ca”. Similarly, we wanted to count “play” and “played” as two tokens under the word-type “play”. When computation is done by hand, the researcher can exercise judgment online to decide whether a particular string of letters should be counted as a word type. Automatic computation, however, is much more literal: Any unique string will be counted as a separate word type. In order to have inflected forms counted as tokens of the uninflected stem (rather than as different word types), we morphemicized inflected forms in transcribing. That is, we transcribed “cats” as “cat-s” and “played” as “play-ed”. Using our morphemicized transcripts, we then instructed FREQ to ignore anything that followed a hyphen (-) within a word. The command string used was:

```
freq +t*CHI +s"*-%%" +f *.cha
```

+t*CHI	Analyze the child speaker only
+s"*-% % "	Ignore the hyphen and subsequent characters
+f	Save output in a file
***.cha**	Analyze all files ending with the extension .cha

The only new element in this command is +s"*-%%". The +s option tells FREQ to search for and count certain strings. Here we ask that, in its search, FREQ ignore any hyphen that occurs within a word, as well as whatever follows the hyphen. In this way, FREQ produces output in which inflected forms of nouns and verbs are not counted as separate word types, but rather as tokens of the uninflected form. The output generated from this analysis goes into five files. For the 68.cha input file, the output is 68.frq.cex. At the end of this file, we find this summary analysis:

```
85      Total number of different word types used
233     Total number of words (tokens)
0.365   Type/Token ratio
```

We can look at each of the five output files to get this summary TTR information for each child.

6.6 Generating Language Profiles

Once we have computed these basic measures of utterance length, lexical diversity, and conversational participation for our five target children, we need to see how each child compares to his or her peers in each of these domains. To do this, we use the means and standard deviations for each measure for the whole New England sample at 20 months, as given in the following table.

Table 5: New England 20 Means

Measure	Mean	SD	Range
MLU50	1.400	0.400	1.02-2.64
MLU5 longest	2.848	1.310	1.00-6.20
TTR	0.433	0.102	0.266-0.621
MLT Ratio	0.246	0.075	0.126-0.453

The distribution of MLU50 scores was quite skewed, with the majority of children who produced at least 50 utterances falling in the MLU range of 1.00-1.20. As noted earlier, 15 of the 48 children failed to produce even 50 utterances. At this age the majority of children in the sample are essentially still at the one-word stage, producing few utterances of more than one word or morpheme. Like MLU50, the shape of the distributions for MLUS and for MLT ratio were also somewhat skewed toward the lower end, though not as severely as was MLU50.

Z-scores, or standard scores, are computed by subtracting each child’s score on a particular measure from the group mean and then dividing the result by the overall standard deviation:

```
(child's score - group mean) ÷ standard deviation
```

The results of this computation are given in the following table.

Table 6: *Z*-scores for Five Children

Child	MLU50	MLU5	TTR	MLT Ratio
14	0.10	0.12	1.84	-0.90
55	-0.70	-0.65	-0.15	-0.94
66	-0.25	-0.19	-0.68	-1.14
68	3.10	2.56	-0.67	1.60
98	-0.95	-1.11	-0.55	0.31

We would not expect to see radical departures from the group means on any of the measures. For the most part, this expectation is borne out: we do not see departures greater than 2 standard deviations from the mean on any measure for any of the five children, except for the particularly high MLU50 and MLU5 observed for Subject 068.

It is not the case, however, that all five of our target children have flat profiles. Some children show marked strengths or weaknesses relative to their peers in particular domains. For example, Subject 14, although very close to the mean in terms of utterance length (MLU5O and MLU5), shows marked strength in lexical diversity (TTR), even though she shoulders relatively little of the conversational burden (as measured by MLT ratio). The strengths of Subject 68, on the other hand, appear to be primarily in the area of syntax (at least as measured by MLU50 and MLU5); her performance on both the lexical and conversational measures (i.e., TTR and MLT ratio) is only mediocre. The subjects at the second and third quartile in terms of MLU (Subject 055 and Subject 066) do have profiles that are relatively flat: Their *z*-scores on each measure fall between -1 and 0. However, the child with the lowest MLU50 (Subject 098) again shows an uneven profile. Despite her limited production, she manages to bear her portion of the conversational load. You will recall that unintelligible vocalizations transcribed as xxx or yyy, as well as nonverbal turns indicated by the postcode [+ trn], are all counted in computing MLT. Therefore, it is possible that many of this child’s turns consisted of unintelligible vocalizations or nonverbal gestures.

What we have seen in examining the profiles for these five children is that, even among normally developing children, different children may have strengths in different domains, relative to their age mates. For illustrative purposes here I have considered only three domains, as measured by four indices. In order to get a more detailed picture of a child’s language production, we might choose to include other indices, or to further refine the measures we use. For example, we might compute TTR based on a particular number of words, or we might time-sample by examining the number of word types and word tokens the child produced in a given number of minutes of mother–child interaction. We might also consider other measures of conversational competence, such as number of child initiations and responses; fluency measures, such as number of retraces or hesitations; or pragmatic measures, such as variety of speech acts produced. Computation of some of these measures would require that codes be entered into the transcript prior to analysis; however, the CLAN analyses themselves would, for the most part, simply be variations on the techniques I have discussed in this chapter. In the exercises that follow, you will have an opportunity to use these techniques to perform analyses on these five children at both 20 months and 32 months.

6.7 Further Exercises

The files needed for the following exercises are in two directories in the /lib folder: NE20 and NE32. No data are available for Subject 14 at 32 months.

1. Compute the length in morphemes of each target child's single longest utterance at 20 months. Compare with the MLU of the five longest utterances. Consider why a researcher might want to use MLU of the five longest rather than MLU of the single longest utterance.
2. Use the +z option to compute TTR on each child's first 50 words at 32 months. Then do the same for each successive 50-word band up to 300. Check the output each time to be sure that 50 words were in fact found. If you specify a range of 50 words where there are fewer than 50 words available in the file, FREQ still performs the analysis, but the output will show the actual number of tokens found. What do you observe about the stability of TTR across different samples of 50 words?
3. Use the MLU and FREQ programs to examine the mother's (*MOT) language to her child at 20 months and at 32 months.What do you observe about the length/complexity and lexical diversity of the mother's speech to her child? Do they remain generally the same across time or change as the child's language develops? If you observe change, how can it be characterized?
4. Perform the same analyses for the four target children for whom data are available at age 32 months. Use the data given earlier to compute *z*-scores for each target child on each measure (MLU 50 utterances, MLU of five longest utterances, TTR, MLT ratio). Then plot profiles for each of the target children at 32 months. What consistencies and inconsistencies do you see from 20 to 32 months? Which children, if any, have similar profiles at both ages? Which children's profiles change markedly from 20 to 32 months?
5. Conduct a case study of a child you know to explore whether type of activity and/or interlocutor affect mean length of turn (MLT). Videotape the child and mother engaged in two different activities (e.g., bookreading, having a snack together, playing with a favorite toy). On another occasion, videotape the child engaged in the same activities with an unfamiliar adult. If it is not possible to videotape, you may audiotape and supplement with contextual notes. Transcribe the interactions in CHAT format. You may wish to put each activity in a separate file (or see CLAN manual for how to use the program GEM). Compare the MLT ratio for each activity and adult–child pair. Describe any differences you observe.

7 Features

7.1 Shell Commands

CLAN provides two types of commands. The first are the Shell commands. These are utility commands like those in the old-style DOS or Unix shells. These commands are available for use inside the **Commands** window. All of the commands except those marked with an asterisk are available for both Macintosh and Windows versions of CLAN. The following commands allow you to change your folder or directory, display information, or launch a new program.

accept*	This command applies only to Macintosh. If you only want to have CLAN look at files that the Macintosh calls TEXT files, then type: accept text. If you want to set this back to all files, type *accept all*.
batch	You can place a group of commands into a text file which you then execute as a batch. The word *batch* should be followed by the name of a file in your working directory. Each line of that file is then executed as a CLAN command.
cd	This command allows you to change directories. With two dots, you can move up one directory. If you type a folder's name and the folder is in the current folder, you can move right to that folder. If you type a folder's absolute address, you can move to that folder from any other folder. For example, the command **cd HardDisk:Applications:**CLAN on the Macintosh will take you to the CLAN directory.
copy	If you want to copy files without going back to the Finder, you can use this command. The -q option asks to make sure you want to make the copy.
del	This command allows you to delete files. Using this in combination with the +re switch can be very dangerous. In this combination, the command *del* * can delete all files from your current working directory and those below it. **Please be careful!**
dir	This command lists all the files in your current directory.
info	This command displays the available programs and commands.
list	This command lists the files that are currently in your input files list.
rmdir	This command deletes a directory or folder.
ren*	This command allows you to change file names in a variety of ways. The rename command can use the asterisk as a wildcard for files in which there is a period. You can change case by using -u for upper and -l for lower. You can change extensions by using wildcards in file names. The -c and -t switches allow you to change the creator signature and file types recognized

by Macintosh. Usually, you will want to have TEXT file types. CLAN produces these by default and you should seldom need to use the -t option. You will find that the -c option is more useful. On the Macintosh, if you want a set of files to have the icon and ownership for CLAN, you should use this command:

```
ren -cMCED *.cha *.cha
```

If you have spaces in these names, surround them with single quotes. For example, to change ownership to the MPW shell, you would need quotes in order to include the additional fourth space character:

```
ren -c'MPS ' *.cha *.cha
```

Or you could rename a series of files with names like "child.CHA (Word 5)," using this command:

```
ren '*.CHA (Word 5)' *.cha
```

type This command displays a file in the CLAN output window.

7.2 Online Help

CLAN has a limited form of online help. To use this help, you simply type the name of the command without any further options and without a file name. The computer will then provide you with a brief description of the command and a list of its available options. To see how this works, just type *freq* and a carriage return and observe what happens. If you need help remembering the various shell commands discussed in the previous section, you can click on the **Help** button at the right of the **Commands** window. If there is something that you do not understand about CLAN, the best thing you can do is to try to find the answer to your problem in this manual.

7.3 Testing CLAN

It is a good idea to make sure that CLAN is conducting analyses correctly. In some cases you may think that the program is doing something different from what it is actually designed to do. In order to prevent misunderstandings and misinterpretations, you should set up a small test file that contains the various features you want CLAN to analyze. For example, if you are running a FREQ analysis, you can set a file with several instances of the words or codes for which you are searching. Be sure to include items that should be "misses" along with those that should be "hits." For example, if you do not want CLAN to count items on a particular tier, make sure you put some unique word on that tier. If the output of FREQ includes that word, you know that something is wrong. In general, you should be testing not for correct performance but for possible incorrect performance. In order to make sure that you are using the +t and +s switches correctly, make up a small file and then run KWAL over it without specifying any +s switch. This should output exactly the parts of the file that you intend to include or exclude.

7.4 Bug Reports

Although CLAN has been extensively tested for years, it is possible that some analyses will provide incorrect results. When this occurs, the first thing to do is to reread the relevant sections of the manual to be sure that you have entered all of your commands correctly. If a rereading of the manual does not solve the problem, then you can send e-mail to macw@cmu.edu to try to get further assistance. In some cases, there may be true "bugs" or program errors that are making correct analyses impossible. Should the program not operate properly, please send e-mail to macw@cmu.edu with the following information:

1. a description of the machine you are using and the operating system you are running,
2. a copy of the file that the program was being run on,
3. the complete command line used when the malfunction occurred,
4. all the results obtained by use of that command, and
5. the date of compilation of your CLAN program, which you can find by clicking on "About CLAN" at the top left of the menu bar on Macintosh or the "Help CLAN" option at the top right of the menu bar for Windows.

Use WinZip or Stuffit to save the input and output files and include them as an e-mail attachment. Please try to create the smallest possible file you can that will still illustrate the bug.

7.5 Feature Requests

CLAN has been designed in response to information we have received from users about the kinds of programs they need for furthering their research. Your input is important, because we are continually designing new commands and improving existing programs. If you find that these programs are not capable of producing the specific type of analysis that you are trying to achieve, contact us and we will do our best to help. Sometimes we can explain ways of using CLAN to achieve your goals. In other cases, it may be necessary to modify the program. Each request must include a simple example of an input file and the output you would like, given this input. Also, please explain how this output will help you in your research. You can address inquiries by email to macw@cmu.edu.

8 Analysis Commands

The analytic work of CLAN is performed by a series of commands that search for strings and compute a variety of indices. These commands are all run from the Commands window. In this section, we will examine each of the commands and the various options that they take. The commands are listed alphabetically. The following table provides an overview of the various CLAN commands. The CHECK program is included here, because it is so important for all aspects of use of CLAN.

CLAN also includes two other major groups of commands. The first group is used to perform morphosyntactic analysis on files by tagging words for their part of speech and detecting grammatical relations. These programs are discussed in Chapter 7. In addition, CLAN includes a large group of Utility commands that will be described in the Chapter 8.

Command	**Page**	**Function**
CHAINS	50	Tracks sequences of interactional codes across speakers.
CHECK	54	Verifies the correct use of CHAT format.
CHIP	57	Examines parent-child repetition and expansion.
COMBO	63	Searches for complex string patterns.
COOCUR	71	Examines patterns of co-occurence between words.
DIST	72	Examines patterns of separation between speech act codes.
DSS	73	Computes the Developmental Sentence Score.
FREQ	81	Computes the frequencies of the words in a file or files.
FREQMERG	91	Combines the outputs of various runs of FREQ.
FREQPOS	91	Tracks the frequencies in various utterance positions.
GEM	92	Finds areas of text that were marked with GEM markers.
GEMFREQ	94	Computes frequencies for words inside GEM markers.
GEMLIST	95	Lists the pattern of GEM markers in a file or files.
KEYMAP	96	Lists the frequencies of codes that follow a target code.
KWAL	97	Searches for word patterns and prints the line.
MAXWD	99	Finds the longest words in a file.
MLT	101	Computes the mean length of turn.
MLU	104	Computes the mean length of utterance.
MODREP	109	Matches the child’s phonology to the parental model.
PHONFREQ	112	Computes the frequency of phonemes in various positions.
RELY	113	Measures reliability across two transcriptions.
STATFREQ	114	Formats the output of FREQ for statistical analysis.
TIMEDUR	116	Uses the numbers in sonic bullets to compute overlaps.
VOCD	116	Computes the VOCD lexical diversity measure.
WDLEN	122	Computes the length of utterances in words.

8.1 CHAINS

CHAINS is used to track sequences of interactional codes. These codes must be entered by hand on a single specified coding tier. In order to test out CHAINS, you may wish to try the file chains.cha that contains the following sample data.

```
@Begin
@Participants:	CHI Sarah Target_child, MOT Carol Mother
*MOT:	sure go ahead [c].
%cod:	$A
%spa:	$nia:gi
*CHI:	can I [c] can I really [c].
%cod:	$A $D. $B.
%spa:	$nia:fp $npp:yq.
%sit:	$ext $why. $mor
*MOT:	you do [c] or you don't [c].
%cod:	$B $C.
%spa:	$npp:pa
*MOT:	that's it [c].
%cod:	$C
%spa:	$nia:pa
@End
```

The symbol [c] in this file is used to delimit clauses. Currently, its only role is within the context of CHAINS. The %cod coding tier is a project-specific tier used to code possible worlds, as defined by narrative theory. The %cod, %sit, and %spa tiers have periods inserted to indicate the correspondence between [c] clausal units on the main line and sequences of codes on the dependent tier.

To change the order in which codes are displayed in the output, create a file called codes.ord. This file could be located in either your working directory or in the \childes\clan\lib directory. CHAINS will automatically find this file. If the file is not found then the codes are displayed in alphabetical order, as before. In the codes.ord file, list all codes in any order you like, one code per line. You can list more codes than could be found in any one file. But if you do not list all the codes, the missing codes will be inserted in alphabetical order. All codes must begin with the $ symbol.

8.1.1 Sample Runs

For our first CHAINS analysis of this sample file, let us look at the %spa tier. If you run the command:

```
chains +t%spa chains.cha
```

you will get a complete analysis of all chains of individual speech acts for all speakers, as in the following output:

```
> chains +t%spa chains.cha
CHAINS +t%spa chains.cha
Mon May 17 13:09:34 1999
CHAINS (04-May-99) is conducting analyses on:
  ALL speaker tiers
```

```
        and those speakers' ONLY dependent tiers matching: %SPA;
****************************************
From file <chains.cha>

Speaker markers:  1=*MOT, 2=*CHI

$nia:fp    $nia:gi     $nia:pa  $npp:pa     $npp:yq       line #
0          1           0          0           0              3
2          0           0          0           2              6
0          0           0          1           0             10
0          0           1          0           0             13

ALL speakers:
        $nia:fp     $nia:gi     $nia:pa    $npp:pa    $npp:yq

# chains    1         1           1           1          1
Avg leng   1.00      1.00        1.00       1.00       1.00
Std dev    0.00      0.00        0.00       0.00       0.00
Min leng   1         1           1          1          1
Max leng   1         1           1          1          1

Speakers *MOT:
          $nia:fp   $nia:gi    $nia:pa     $npp:pa     $npp:yq
# chains    0         1           1           1           0
Avg leng   0.00      1.00        1.00       1.00        0.00
Std dev    0.00      0.00        0.00       0.00        0.00
Min leng   0         1           1          1           0
Max leng   0         1           1          1           0
SP Part.   0         1           1          1           0
SP/Total   0.00      1.00        1.00       1.00        0.00

Speakers *CHI:
          $nia:fp   $nia:gi     $nia:pa    $npp:pa     $npp:yq
# chains  1         0           0          0            1
Avg leng  1.00      0.00        0.00       0.00         1.00
Std dev   0.00      0.00        0.00       0.00         0.00
Min leng  1         0           0          0            1
Max leng  1         0           0          0            1
SP Part.  1         0           0          0            1
SP/Total  1.00      0.00        0.00       0.00         1.00
```

It is also possible to use the +s switch to merge the analysis across the various speech act codes. If you do this, alternative instances will still be reported, separated by commas. Here is an example:

```
chains +d +t%spa chains.cha +s$nia:%
```

This command should produce the following output:

```
Speaker markers:  1=*MOT, 2=*CHI

$nia:                                   line #
1 gi                                       3
2 fp                                       6
                                           6
1 pa                                      13
```

```
ALL speakers:
          $nia:
# chains  2
Avg leng  1.50
Std dev   0.50
Min leng  1
Max leng  2

Speakers *MOT:
          $nia:
# chains  2
Avg leng  1.00
Std dev   -0.00
Min leng  1
Max leng  1
SP Part.  2
SP/Total  0.67

Speakers *CHI:
          $nia:

# chains  1
Avg leng  1.00
Std dev   0.00
Min leng  1
Max leng  1
SP Part.  1
SP/Total  0.33
```

You can use CHAINS to track two coding tiers at a time. For example, one can look at chains across both the %cod and the %sit tiers by using the following command. This command also illustrates the use of the +c switch, which allows the user to define units of analysis lower than the utterance. In the example file, the [c] symbol is used to delimit clauses. The following command makes use of this marking:

```
chains +c"[c]" +d +t%cod chains.cha +t%sit
```

The output from this analysis is:

```
Speaker markers: 1=*MOT, 2=*CHI
$a              $b              $c              $d              line #
1                                                                   3
2 $ext $why                                     2 $ext $why         6
                2 $mor                                              6
                1               1                                  11
                               1                                  14

ALL speakers:
          $a              $b              $c              $d
# chains   1               1               1               1
Avg leng  2.00            2.00            2.00            1.00
Std dev   0.00            0.00            0.00            0.00
Min leng  2               2               2               1
Max leng  2               2               2               1
```

```
Speakers *MOT:
          $a              $b              $c              $d
# chains   1               1               1               0
Avg leng  1.00            1.00            2.00            0.00
Std dev   0.00            0.00            0.00            0.00
Min leng  1               1               2               0
Max leng  1               1               2               0
SP Part.  1               1               1               0
SP/Total  0.50            0.50            1.00            0.00

Speakers *CHI:
          $a              $b              $c              $d
# chains  1                1               0               1
Avg leng  1.00            1.00            0.00            1.00
Std dev   0.00            0.00            0.00            0.00
Min leng  1               1               0               1
Max leng  1               1               0               1
SP Part.  1               1               0               1
SP/Total  0.50            0.50            0.00            1.00
```

8.1.2 Unique Options

At the end of our description of each CLAN command, we will list the options that are unique to that command. The commands also use several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

+c The default unit for a CHAINS analysis is the utterance. You can use the +c option to track some unit type other than utterances. The other unit type must be delimited in your files with some other punctuation symbol that you specify after the +c, as in +c"[c]" which uses the symbol [c] as a unit delimiter. If you have a large set of delimiters you can put them in a file and use the form +c@filename. To see how this switch operates try out this command:

```
chains +c"[c]" +d +t%cod chains.cha
```

+d Use this switch to change zeroes to spaces in the output. The following command illustrates this option:

```
chains +d +t%spa chains.cha +s$nia:%
```

The +d1 value of this option works the same as +d, while also displaying every input line in the output.

+sS This option is used to specify particular codes to track. For example, +s$b will track only the $b code. A set of codes to be tracked can be placed in a file and tracked using the form +s@filename. In the examples given earlier, the following command was used to illustrate this feature:

```
chains +d +t%spa chains.cha +s$nia:%
```

+wN Sets the width between columns to N characters.

8.2 CHECK

Checking the syntactic accuracy of a file can be done in two ways. One method is to work within the editor. In the editor, you can start up the CHECK program by just typing *Esc-L*. Alternatively, you can run CHECK as a separate program. The CHECK program checks the syntax of the specified CHAT files. If errors are found, the offending line is printed, followed by a description of the problem.

8.2.1 How CHECK Works

CHECK makes two passes through each CHAT file. On the first pass it checks the overall structure of the file. It makes sure that the file begins with @Begin and ends with @End, that each line starts with either *, @, %, or a tab, and that colons are used properly with main lines, dependent tiers, and headers that require entries. If errors are found at this level, CHECK reports the problem and stops, because further processing would be misleading. If there are problems on this level, you will need to fix them before continuing with CHECK. Errors on the first level can mask the detection of further errors on the second level. It is important not to think that a file has passed CHECK until all errors have been removed.

The second pass checks the detailed structure of the file. To do this, it relies heavily on depfile.cut, which we call the "depfile." The depfile distributed with CLAN lists the legitimate CHAT headers and dependent tier names as well as many of the strings allowed within the main line and the various dependent tiers. When running CHECK, you should have the file called depfile.cut located in your LIB directory, which you set from the Commands window. If the programs cannot find the depfile, they will query you for its location.

To get an idea of how CHECK operates, open up the file kid10.cha in the library directory. That file has a large number of CHAT errors. Type *Esc-L*. Try to fix the errors. If you can put the file into correct CHAT format so that it passes cleanly through CHECK, you will have learned how to use CHECK to verify CHAT format.

If you find that the depfile is not permitting things that are important to your research, please contact macw@cmu.edu to discuss ways in which we can extend the CHAT system and its reflection in the XML Schema.

8.2.2 CHECK in CA Mode

CHECK can also be used with files that have been produced using CA mode. The features that CHECK is looking for in CA Mode are:

1. Each utterance should begin with a number and a speaker code in the form #:speaker:<whitespace>.
2. There should be paired parentheses around pause numbers.
3. Numbers marking pause duration are allowed on their own line.
4. Latching should be paired.
5. The double parentheses marking comments should be paired.

6. Overlap markers should be paired.
7. Superscript zeros should be paired.
8. The up-arrow, down-arrow, and zeros are allowed inside words.

8.2.3 Running CHECK

There are two ways to run CHECK. If you are working on new data, it is easiest to run CHECK from inside the editor. To do this, you type *Esc-L* and check runs through the file looking for errors. It highlights the point of the error and tells you what the nature of the error is. Then you need to fix the error in order to allow CHECK to move on through the file.

The other way of running CHECK is to issue the command from the commands window. This is the best method to use when you want to check a large collection of files. If you want to examine several directories, you can use the +re option to make check work recursively across directories. If you send the output of check to the **CLAN Output** window, you can locate errors in that window and then triple-click on the file name and CLAN will take you right to the problem that needs to be fixed. This is an excellent way of working when you have many files and only a few errors.

8.2.4 Restrictions on Word Forms

In order to guarantee consistent transcription of word forms and to facilitate the building of MOR grammars for various languages, CHAT has adopted a set of tight restrictions on word forms. Earlier versions of CLAN and CHAT were considerably less restrictive. However, this absence of tight rules led to many inaccuracies in transcription and analysis. Beginning in 1998, the rules were significantly tightened. In addition, an earlier system of marking morphemes on the main line was dropped in favor of automatic analysis of words through MOR. The various options for word level transcription are summarized in the chapter of the CHAT manual on Words. However, it is useful here to provide some additional detail regarding specific CHECK features.

One major restriction on words forms is that they cannot include numbers. Earlier versions of CHAT allowed for numbers inside UNIBET representations. However, since we now use IPA instead of UNIBET for phonological coding, numbers are no longer needed in this context. Also, actual numbers such as “79” are written out in their component words as “seventy nine” without dashes. Therefore numbers are not needed in this context either.

We also do not allow capital letters inside words. This is done to avoid errors and forms that cannot be recognized by MOR. The exceptions to this principle are for words with underlining, as in Santa_Claus or F_B_I.

CHECK also prohibits dashes within words in many contexts. Dashes were eliminated from most contexts in 1998, but are still available for certain special cases. If there is no an '@' sysmbol before the dash then:

a. it is legal if the word starts with a capital letter
b. it is legal if Legacy used in @Languages.
c. otherwise it is always illegal.

If there is an '@' symbol before the dash then:

a. it is legal if an @u used, i.e. Word-_#~'@u
b. it is illegal if any following symbol used: ,'.!?_-+~#@)"
c. if it is at the end of the word, i.e word-

8.2.5 Some Hints

1. Use CHECK early and often, particularly when you are learning to code in CHAT. When you begin transcribing, check your file inside the editor using *Esc-L*, even before it is complete. When CHECK complains about something, you can learn right away how to fix it before continuing with the same error.
2. If you are being overwhelmed by CHECK errors, you can use the +d1 switch to limit error reports to one of each type. Or you can focus your work first on eliminating main line errors by using the -t% switch.
3. Learn how to use the query-replace function in your text editor to make general changes and CHSTRING to make changes across sets of files.

8.2.6 Unique Options

+d This option attempts to suppress repeated warnings of the same error type. It is convenient to use this in your initial runs when your file has consistent repeated divergences from standard CHAT form. However, you must be careful not to rely too much on this switch, because it will mask many types of errors you will eventually want to correct. The +d1 value of this switch represses errors even more severely to only one of each type.

+e This switch allows the user to select a particular type of error for checking. To find the numbers for the different errors, type:

```
check +e
```

Then look for the error type you want to track, such as error #16, and type:

```
check +e16 *.cha
```

+g1 Setting +g1 turns on the treatment of prosodic contour markers such as -. or -? as utterance delimiters, as discussed in the section on prosodic delimiters in the CHAT manual. Setting -g1 sets the treatment back to the default, which is to not treat these codes as delimiters.

+g2 By default, CHECK requires tabs after the colon on the main line and at the beginning of each line. However, versions of Word Perfect before 5.0 cannot write out text files that include tabs. Other non-ASCII editors may also have this problem. To get around the problem, you can set the -g2 switch in CHECK that stops checking for tabs. If you want to turn this type of checking back on, use the +g2 switch.

+g3 Without the +g3 switch, CHECK does minimal checking for the correctness of the internal contents of words. With this switch turned on, the program makes sure that words do not contain numbers, capital letters, or spurious apostrophes.

CHECK also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.3 CHIP

CHIP was designed and written by Jeffrey Sokolov. The program analyzes specified pairs of utterances. CHIP has been used to explore parental input, the relation between speech acts and imitation, and individual differences in imitativeness in both normal and language-impaired children. Researchers who publish work based on the use of this program should cite Sokolov and MacWhinney (1990). There are four major aspects of CHIP to be described: (1) the tier creation system, (2) the coding system, (3) the technique for defining substitution classes, and (4) the nature of the summary statistics.

8.3.1 The Tier Creation System

CHIP compares two specified utterances and produces an analysis that it then inserts onto a new coding tier. The first utterance in the designated utterance pair is the “source” utterance and the second is the “response” utterance. The response is compared to the source. Speakers are designated by the +b and +c codes. An example of a minimal CHIP command is as follows:

```
chip +bMOT +cCHI chip.cha
```

We can run this command runs on the following seven-utterance chip.cha file that is distributed with CLAN.

```
@Begin
@Participants:  MOT Mother, CHI Child
*MOT: what-'is that?
*CHI: hat.
*MOT: a hat!
*CHI: a hat.
*MOT: and what-'is this?
*CHI: a hat !
*MOT: yes that-'is the hat .
@End
```

The output from running this simple CHIP command on this short file is as follows:

```
CHIP (04-May-99) is conducting analyses on:
  ALL speaker tiers
****************************************
From file <chip.cha>
*MOT:  what-'is that ?
*CHI:  hat .
%chi:  $NO_REP $REP = 0.00
*MOT:  a hat !
%asr:  $NO_REP $REP = 0.00
%adu:  $EXA:hat $ADD:a $EXPAN $DIST = 1 $REP = 0.50
```

```
*CHI:	a hat .
%csr:	$EXA:hat $ADD:a $EXPAN $DIST = 2 $REP = 0.50
%chi:	$EXA:a-hat $EXACT $DIST = 1 $REP = 1.00
*MOT:	and what-'is this ?
%asr:	$NO_REP $REP = 0.00
%adu:	$NO_REP $REP = 0.00
*CHI:	that a hat !
%csr:	$EXA:a-hat $ADD:that $EXPAN $DIST = 2 $REP = 0.67
%chi:	$NO_REP $REP = 0.00
*MOT:	yes that-'is the hat .
%asr:	$NO_REP $REP = 0.00
%adu:	$EXA:that  $EXA:hat  $ADD:yes  $ADD:the  $DEL:a  $MADD:-'is
$DIST $REP = 0.50
```

The output also includes a long set of summary statistics which are discussed later. In the first part of this output, CHIP has introduced four different dependent tiers:

> **%chi:** This tier is an analysis of the child's response to an adult's utterance, so the adult's utterance is the source and the child's utterance is the response.
> **%adu:** This tier is an analysis of the adult's response to a child's utterance, so the child is the source and the adult is the response.
> **%csr:** This tier is an analysis of the child's self repetitions. Here the child is both the source and the response.
> **%asr:** This tier is an analysis of the adult's self repetitions. Here the adult is both the source and the response.

By default, CHIP produces all four of these tiers. However, through the use of the -n option, the user can limit the tiers that are produced. Three combinations are possible:

1. You can use both -ns and -nb. The -ns switch excludes both the %csr tier and the %asr tier. The -nb switch excludes the %adu tier. Use of both switches results in an analysis that computes only the %chi tier.
2. You can use both -ns and -nc. The -ns switch excludes both the %csr tier and the %asr tier. The -nc switch excludes the %chi tier. Use of both of these switches results in an analysis that computes only the %adu tier.
3. You can use both -nb and -nc. This results in an analysis that produces only the %csr and the %asr tiers.

It is not possible to use all three of these switches at once.

8.3.2 The Coding System

The CHIP coding system includes aspects of several earlier systems (Bohannon & Stanowicz, 1988; Demetras, Post, & Snow, 1986; Hirsh-Pasek, Trieman, & Schneiderman, 1984; Hoff-Ginsberg, 1985; Moerk, 1983; Nelson, Denninger, Bonvilian, Kaplan, & Baker, 1984). It differs from earlier systems in that it computes codes automatically. This leads to increases in speed and reliability, but certain decreases in flexibility and coverage.

The codes produced by CHIP indicate lexical and morphological additions, deletions, exact matches and substitutions. The codes are as follows:

$ADD	additions of N continuous words
$DEL	deletions of N continuous words
$EXA	exact matches of N continuous words
$SUB	substitutions of N continuous words from within a word list
$MADD	morphological addition based on matching word stem
$MDEL	morphological deletion based on matching word stem
$MEXA	morphological exact match based on matching word stem
$MSUB	morphological substitution based on matching word stem
$DIST	the distance the response utterance is from the source
$NO_REP	the source and response do not overlap
$LO_REP	the overlap is below a user-specified minimum
$EXACT	source-response pairs with no changes
$EXPAN	pairs with additions but no deletions or substitutions
$REDUC	pairs with deletions but no additions or substitutions
$SUBST	source-response pairs with only exact-matches and substitutions
$FRO	an item from the word list has been fronted
$REP	the percentage of repetition between source and response

Let us take the last line of the chip.cha file as an example:

```
*MOT:   yes that-'is the hat .
%asr:   $NO_REP $REP = 0.00
%adu:   $EXA:hat $ADD:yes-that-'is-the $DEL:a $DIST = 1 $REP = 0.25
```

The %adu dependent tier indicates that the adult's response contained an EXAct match of the string "hat," the ADDition of the string "yes-that-'is-the" and the DELetion of "a." The DIST=1 indicates that the adult's response was "one" utterance from the child's, and the repetition index for this comparison was 0.25 (1 matching stem divided by 4 total stems in the adult's response).

CHIP also takes advantage of CHAT-style morphological coding. Upon encountering a word, the program determines the word's stem and then stores any associated prefixes or suffixes along with the stem. During the coding process, if lexical stems match exactly, the program then also looks for additions, deletions, repetitions, or substitutions of attached morphemes.

8.3.3 Word Class Analysis

In the standard analysis of the last line of the chip.cha file, the fact that the adult and the child both use a definite article before the noun *hat* is not registered by the default CHIP analysis. However, it is possible to set up a substitution class for small groups of words such as definite articles or modal auxiliaries that will allow CHIP to track such within-class substitutions, as well as to analyze within-class deletions, additions, or exact repetitions. To do this, the user must first create a file containing the list of words to be considered as substitutions. For example to code the substitution of articles, the file distributed with CLAN called articles.cut can be used. This file has just the two articles *a* and *the*. Both the +g option and the +h (word-list file name) options are used, as in the following example:

```
chip +cCHI +bMOT +g +harticles.cut chip.cha
```

The output of this command will add a $SUB field to the %adu tier:

```
*CHI:   a hat!
*MOT:   yes that-'is the hat.
%adu:   $EXA:that $EXA:hat $ADD:yes $SUB:the $MADD:-'is $DIST = 1
$REP =0.50
```

The +g option enables the substitutions, and the +harticle.cut option directs CHIP to examine the word list previously created by the user. Note that the %adu now indicates that there was an EXAct repetition of *hat*, an ADDition of the string *yes that-'is* and a within-class substitution of *the* for *a*. If the substitution option is used, EXPANsions and REDUCtions are tracked for the included word list only. In addition to modifying the dependent tier, using the substitution option also affects the summary statistics that are produced. With the substitution option, the summary statistics will be calculated relative only to the word list included with the +h switch. In many cases, you will want to run CHIP analyses both with and without the substitution option and compare the contrasting analyses.

You can also use CLAN iterative limiting techniques to increase the power of your CHIP analyses. If you are interested in isolating and coding those parental responses that were expansions involving closed-class verbs, you would first perform a CHIP analysis and then use KWAL to obtain a smaller collection of examples. Once this smaller list is obtained, it may be hand coded and then once again submitted to KWAL or FREQ analysis. This notion of iterative analysis is extremely powerful and takes full advantage of the benefits of both automatic and manual coding.

8.3.4 Summary Measures

In addition to analyzing utterances and creating separate dependent tiers, CHIP also produces a set of summary measures. These measures include absolute and proportional values for each of the coding categories for each speaker type that are outlined below. The definition of each of these measures is as follows. In these codes, the asterisk stands for any one of the four basic operations of ADD, DEL, EXA, and SUB.

Total # of Utterances	The number of utterances for all speakers regardless of the number of intervening utterances and speaker identification.
Total Responses	The total number of responses for each speaker type regardless of amount of overlap.
Overlap	The number of responses in which there is an overlap of at least one word stem in the source and response utterances.

No Overlap	The number of responses in which there is NO overlap between the source and response utterances.
Avg_Dist	The sum of the DIST values divided by the total number of overlapping utterances.
%_Overlap	The percentage of overlapping responses over the total number of responses.
Rep_Index	Average proportion of repetition between the source and response utterance across all the overlapping responses in the data.
*_OPS	The total (absolute) number of add, delete, exact, or substitution operations for all overlapping utterance pairs in the data.
%_*_OPS	The numerator in these percentages is the operator being tracked and the denominator is the sum of all four operator types.
*_WORD	The total (absolute) number of add, delete, exact, or substitution words for all overlapping utterance pairs in the data.
%_*_WORDS	The numerator in these percentages is the word operator being tracked and the denominator is the sum of all four word operator types.
MORPH_*	The total number of morphological changes on exactlymatching stems.
%_MORPH_*	The total number of morphological changes divided by the number of exactly matching stems.
AV_WORD_*	The average number of words per operation across all the overlapping utterance pairs in the data.
FRONTED	The number of lexical items from the word list that have been fronted.
EXACT	The number of exactly matching responses.
EXPAN	The number of responses containing only exact matches and additions.

REDUC	The number of responses containing only exact-matches and deletions.
SUBST	The number of responses containing only exact matches and substitutions.

8.3.5 Unique Options

+b Specify that speaker ID S is an “adult.” The speaker does not actually have to be an adult. The “b” simply indicates a way of keeping track of one of the speakers.

+c Specify that speaker ID S is a “child.” The speaker does not actually have to be a child. The “c” simply indicates a way of keeping track of one of the speakers.

+d Using +d with no further number outputs only coding tiers, which are useful for iterative analyses. Using +d1 outputs only summary statistics, which can then be sent to a statistical program.

+g Enable the substitution option. This option is meaningful in the presence of a word list in a file specified by the +h/-h switch, because substitutions are coded with respect to this list.

+h Use a word list file. The target file is specified after the letter “h.” Words to be included (with +h) or excluded (with -h) are searched for in the target file. The use of an include file enables CHIP to compare ADD and DEL categories for any utterance pair analyses to determine if there are substitutions within word classes. For example, the use of a file containing a list of pronouns would enable CHIP to determine that the instances of ADD of “I” and DEL of “you” across a source and response utterance are substitutions within a word class.

Standard CLAN wildcards may be used anywhere in the word list. When the transcript uses CHAT-style morphological coding (e.g., I-’ve), only words from the word list file will match to stems in the transcript. In other words, specific morphology may not be traced within a word list analysis. Note that all of the operation and word-based summary statistics are tabulated with respect to the word list only. The word list option may be used for any research purpose including grammatical word classes, number terms, color terms, or mental verbs. Note also that the -h option is useful for excluding certain terms such as “okay” or “yeah” from the analysis. Doing this often improves the ability of the program to pick up matching utterances.

+n This switch has three values: +nb, +nc, and +ns. See the examples given earlier for a discussion of the use of these switches in combination.

+qN Set the utterance window to N utterances. The default window is seven utterances. CHIP identifies the source-response utterances pairs to code. When a response is encountered, the program works backwards (through a window determined by the +q option) until it identifies the most recent potential source utterance. Only one source utterance is coded for each response utterance. Once the source-response pair has been identified, a simple matching procedure is performed.

+x Set the minimum repetition index for coding.

CHIP also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.4 COMBO

COMBO provides the user with ways of composing Boolean search strings to match patterns of letters, words, or groups of words in the data files. This program is particularly important for researchers who are interested in syntactic analysis. The search strings are specified with either the +s/-s option or in a separate file. Use of the +s switch is obligatory in COMBO. When learning to use COMBO, what is most tricky is learning how to specify the correct search strings.

8.4.1 Composing Search Strings

Boolean searching uses algebraic symbols to better define words or combinations of words to be searched for in data. COMBO uses regular expressions to define the search pattern. These six special symbols are listed in the following table:

Table 3: COMBO Strings

Meaning	**Type**	**Symbol**
immediately FOLLOWED by	Boolean	^
inclusive OR	Boolean	+
logical NOT	Boolean	!
repeated character	metacharacter	*
single character	metacharacter	_
quoting	metacharacter	\

Inserting the ^ operator between two strings causes the program to search for the first string followed by the second string. The + operator inserted between two strings causes the program to search for either of the two strings. In this case, it is not necessary for both of them to match the text to have a successful match of the whole expression. Any one match is sufficient. The ! operated inserted before a string causes the program to match a string of text that does not contain that string.

The items of the regular expression will be matched to the items in the text *only* if they directly follow one another. For example, the expression **big^cat** will match only the word *big* directly followed by the word *cat* as in *big cat*. To find the word *big* followed by the word *cat* immediately or otherwise, use the metacharacter * between the items *big* and *cat*, as in **big^*^cat**. This expression will match, for example, *big black cat*. Notice that, in this example, * ends up matching not just any string of characters, but any string of words or characters up to the point where *cat* is matched. Inside a word, such as *go**, the asterisk stands for any number of characters. In the form ^*^, it stands for any number of words. The * alone cannot be used in conjunction with the +g or +x option.

The underscore is used to "stand in for" for any *single* character. If you want to match any single word, you can use the underscore with the asterisk as in +s"_*." which will match any single word followed by a period. For example, in the string *cat.*, the underscore would match *c*, the asterisk would match *at* and the period would match the period.

The backslash (\) is used to quote either the asterisk or the underline. When you want to search for the actual characters * and _, rather than using them as metacharacters, you insert the \ character before them.

Using metacharacters can be quite helpful in defining search strings. Suppose you want to search for the words *weight, weighs, weighing, weighed,* and *weigh*. You could use the string *weigh** to find all of the previously mentioned forms. Metacharacters may be used anywhere in the search string.

When COMBO finds a match to a search string, it prints out the entire utterance in which the search string matched, along with any previous context or following context that had been included with the +w or -w switches. This whole area printed out is what we will call the "window."

8.4.2 Examples of Search Strings

The following command searches the sample.cha file and prints out the window which contains the word "want" when it is directly followed by the word "to."

```
combo +swant^to sample.cha
```

If you are interested not just in cases where "to" immediately follows "want," but also cases where it eventually follows, you can use the following command syntax:

```
combo +s"want^*^to" sample.cha
```

The next command searches the file and prints out any window that contains both "want" and "to" in any order:

```
combo +s"want^to" +x sample.cha
```

The next command searches sample.cha and sample2.cha for the words "wonderful" or "chalk" and prints the window that contains either word:

```
combo +s"wonderful+chalk" sample*.cha
```

The next command searches sample.cha for the word "neat" when it is *not* directly followed by the words "toy" or "toy-s." Note that you need the ^ in addition to the ! in order to clearly specify the exact nature of the search you wish to be performed.

```
combo +s"neat^!toy*" sample.cha
```

In this next example, the COMBO program will search the text for either the word “see” directly followed by the word “what” or all the words matching “toy*.”

```
combo +s"see^(what+toy*)" sample.cha
```

You can use parentheses in order to group the search strings unambiguously as in the next example:

```
combo +s"what*^(other+that*)" sample.cha
```

This command causes the program to search for words matching “what” followed by either the word “that” or the word “other.” An example of the types of strings that would be found are: “what that,” “what’s that,” and “what other.” It will not match “what is that” or “what do you want.” Parentheses are necessary in the command line because the program reads the string from left to right. Parentheses are also important in the next example.

```
combo +s"the^*^!grey^*^(dog+cat)" sample2.cha
```

This command causes the program to search the file sample2.cha for *the* followed, immediately or eventually, by any word or words except *grey*. This combination is then to be followed by either *dog* or *cat*. The intention of this search is to find strings like *the big dog* or *the boy with a cat*, and not to match strings like *the big grey cat*. Note the use of the parentheses in the example. Without parentheses around *dog+cat*, the program would match simply *cat*. In this example, the sequence `^*^` is used to indicate “immediately or later.” If we had used only the symbol `^` instead of the `^*^`, we would have matched only strings in which the word immediately following *the* was not *grey*.

8.4.3 Referring to Files in Search Strings

Inside the +s switch, one can include reference to one, two, or even more groups of words that are listed in separate files. For example, you can look for combinations of prepositions with articles by using this switch:

```
+s@preps^@arts
```

To use this form, you first need to create a file of prepositions called “preps” with one preposition on each line and a file of articles called “arts” with one article on each line. By maintaining files of words for different parts of speech or different semantic fields, you can use COMBO to achieve a wide variety of syntactic and semantic analyses. Some suggestions for words to be grouped into files are given in the chapter of the CHAT manual on word lists. Some particularly easy lists to create would be those including all the modal verbs, all the articles, or all the prepositions. When building these lists, remember the possible existence of dialect and spelling variations such as *dat* for *that*.

Here is a somewhat more complex example of how to refer to files in search strings. In this case, we are looking in Spanish files for words that follow the definite articles *la* and *el* and begin with either vowels or the silent “h” followed by a vowel. So we can have one file, called arts.cut, with the words el and la each on their own line. Then, we can have another file, called vowels.cut, that looks like this:

```
hu*
u*
ha*
a*    etc.
```

In this case, the command we use looks like this:

```
combo +s@arts.cut^@vowels.cut test.cha
```

8.4.4 Cross-tier Combo

Particular dependent tiers can be included or excluded by using the +t option immediately followed by the tier code. By default, COMBO excludes the header and dependent code tiers from the search and output. However, when the dependent code tiers are included by using the +t option, they are combined with their speaker tiers into clusters. For example, if the search expression is **the^*^kitten**, the match would be found even if *the* is on the speaker tier and *kitten* is on one of the speaker's associated dependent tiers. This feature is useful if one wants to select for analyses only speaker tiers that contain specific word(s) on the main tier and some specific codes on the dependent code tier. For example, if one wants to produce a frequency count of the words *want* and *to* when either one of them is coded as an imitation on the %spa line, or *neat* when it is a continuation on the %spa line, the following two commands could be used:

```
combo +s(want^to^*^%spa:^*^$INI*)+(neat^*^%spa:^*^$CON*)
       +t%spa +f +d sample.cha
freq +swant +sto +sneat sample.cmb
```

In this example, the +s option specifies that the words *want, to,* and *$INI* may occur in any order on the selected tiers. The +t%spa option must be added in order to allow the program to look at the %spa tier when searching for a match. The +d option is used to specify that the information produced by the program, such as file name, line number and exact position of words on the tier, should be excluded from the output. This way the output is in a legal CHAT format and can be used as an input to another CLAN program, FREQ in this case. The same effect could also be obtained by using the piping feature.

8.4.5 Cluster Pairs in COMBO

Most computer search programs work on a single line at a time. If these programs find a match on the line, they print it out and then move on. Because of the structure of CHAT and the relation between the main line and the dependent tiers, it is more useful to have the CLAN programs work on "clusters" instead of lines. The notion of a cluster is particularly important for search programs, such as COMBO and KWAL. A cluster can be defined as a single utterance by a single speaker, along with all of its dependent tiers. By default, CLAN programs work on a single cluster at a time. For COMBO, one can extend this search scope to a pair of contiguous clusters by using the +b switch. However, this switch should only be used when cross-cluster matches are important, because addition of the switch tends to slow down the running of the program. To illustrate the use of the +b switch, consider how you might want to perform a FREQ analysis on sentences that the mother directs to the younger child, as opposed to sentences directed to the older child or other adults. To find the sentences directed to the younger child, one can imagine that sentences from the mother that are followed by sentences from the younger child are most likely directed to the younger child. To find these, you can use this command:

```
Combo +b2 +t*MOT +t*CHI +s\*MOT:^*^\*CHI: eve01.cha
```

After having checked out the results of this basic command, you can then pipe the data to FREQ using this full command:

```
Combo +b2 +t*MOT +t*CHI +s\*MOT:^*^\*CHI: eve01.cha +d | freq
+t*MOT
```

8.4.6 Searching for Clausemates

When conducting analyses on the %syn tier, researchers often want to make sure that the matches they locate are confined to “clausemate” constituents. Consider the following two %syn tiers:

```
%syn:  ( S V L ( O V ) )
%syn:  ( S V ( S V O ) )
```

If we want to search for all subjects (S) followed by objects (O), we want to make sure that we match only patterns of the type found in the embedded clause in the second example. If we use a simple search pattern such as +sS^*^O", we will match the first example as well as both clauses in the second example. In order to prevent this, we need to add parentheses checking to our search string. The string then becomes:

```
+s"S^*^(!\(+!\))^*^O
```

This will find only subjects that are followed by objects without intervening parentheses. In order to guarantee the correct detection of parentheses, they must be surrounded by spaces on the %syn line.

8.4.7 Tracking Final Words

In order to find the final words of utterances, you need to use the complete delimiter set in your COMBO search string. You can do this with this syntax (\!+?+.) where the parentheses enclose a set of alternative delimiters. In order to specify the single word that appears before these delimiters, you can use the asterisk wildcard preceded by an underline. Note that this use of the asterisk treats it as referring to any number of letters, rather than any number of words. By itself, the asterisk in COMBO search strings usually means any number of words, but when preceded by the underline, it means any number of characters. Here is the full command:

```
combo +s"_*^(\!+?+.)" sample.cha
```

This can then be piped to FREQ if the +d3 switch is used:

```
combo +s"_*^(\!+?+.)" +d3 sample.cha | freq
```

8.4.8 Tracking Initial Words

Because there is no special character that marks the beginnings of files, it is difficult to compose search strings to track items at utterance initial position. To solve this problem, you can run use CHSTRING to insert sentence initial markers. A good marker to use is the ++ symbol, which is only rarely used for other purposes. You can use this command:

```
chstring +c -t@ -t% +t* *.cha
```

You also need to have a file called changes.cut that has this one line:

```
":              "    ":       ++"
```

In this one-line file, there are two quoted strings. The first has a colon followed by a tab; the second has a colon followed by a tab and then a double plus.

8.4.9 Adding Excluded Characters

COMBO strings have no facility for excluding a particular set of words. However, you can achieve this same effect by (1) matching a pattern, (2) outputting the matches in CHAT format, (3) altering unwanted matches so they will not rematch, and (4) then rematching with the original search string. Here is an example:

```
combo +s"*ing*" +d input.cha | chstring +c +d -f | combo +s"*ing*"
```

The goal of this analysis is to match only words ending in participial *ing*. First, COMBO matches all words ending in *ing*. Then CHSTRING takes a list of unwanted words that end in *ing* like *during* and *thing* and changes the *ing* in these words to *iing*, for example. Then COMBO runs again and matches only the desired participial forms.

8.4.10 Limiting with COMBO

Often researchers want to limit their analysis to some particular group of utterances. CLAN provides the user with a series of switches within each program for doing the simplest types of limiting. For example, the +t/-t switch allows the user to include or exclude whole tiers. However, sometimes these simple mechanisms are not sufficient and the user will have to use COMBO or KWAL for more detailed control of limiting. COMBO is the most powerful program for limiting, because it has the most versatile methods for string search using the +s switch. Here is an illustration. Suppose that, in sample.cha, you want to find the frequency count of all the speech act codes associated with the speaker *MOT when this speaker used the phrase “want to” in an utterance. To accomplish this analysis, use this command:

```
combo +t*MOT +t%spa sample.cha +s"want^to" +d | freq
```

The +t*MOT switch (Unix users should add double quotes for +t"*MOT") tells the program to select only the main lines associated with the speaker *MOT. The +t%spa tells the program to add the %spa tier to the *MOT main speaker tiers. By default, the dependent tiers are excluded from the analysis. Then follows the file name, which can appear anywhere after the program name. The +s"want^to" then tells the program to select only the *MOT clusters that contain the phrase *want to*. The +d option tells the program to output the matching clusters from sample.cha without any non-CHAT identification information. Then the results are sent through a “pipe” indicated by the | symbol to the FREQ program, which conducts an analysis on the main line. The results could also be piped on to other programs such as MLU or KEYMAP or they can be stored in files.

Sometimes researchers want to maintain a copy of their data that is stripped of the various coding tiers. This can be done by this command:

```
combo +s* +o@ -t% +f *.cha
```

The +o switch controls the addition of the header material that would otherwise be excluded from the output and the -t switch controls the deletion of the dependent tiers. It is also possible to include or exclude individual speakers or dependent tiers by providing additional +t or -t switches. The best way to understand the use of limiting for controlling data display is to try the various options on a small sample file.

8.4.11 Adding Codes with COMBO

Often researchers leave a mark in a transcript indicating that a certain sentence has matched some search pattern. For example, imagine that you want to locate all sentences with a preposition followed immediately by the word "the" and then tag these sentences in some way. You can use the COMBO +d4 switch to do this. First, you would create a file with all the prepositions (one on each line) and call it something like prep.cut. Then you would create a second support file called something like combo.cut with this line:

```
"@prep.cut^the" "$Pthe" "%cod:"
```

The first string in this line gives the term used by the standard +s search switch. The second string says that the code produced will bye $Pthe. The third string says that this code should be placed on a %cod line under the utterance that is matched. If there is no %cod line there yet, one will be created. The COMBO command that uses this information would then be:

```
combo +s"@combo.cut" +d4 filename.cha
```

The resulting file will have this line added:

```
%cod:	$Pthe
```

You can include as many lines as you wish in the combo.cut file to control the addition of additional codes and additional coding lines. Once you are done with this, you can use these new codes to control better inclusion and exclusion of utterances and other types of searches.

8.4.12 Unique Options

+b COMBO usually works on only one cluster at a time. However, when you want to look at a contiguous pair of clusters, you can use this switch.

+d Normally, COMBO outputs the location of the tier where the match occurs. When the +d switch is turned on you can output only each matched sentence in a simple legal CHAT format. The +d1 switch outputs legal CHAT format along with line numbers and file names. The +d2 switch outputs files names once per file only. The +d3 switch outputs legal CHAT format, but with only the actual words matched by the search string, along with @Comment headers that are ignored by other programs. Try these commands:

```
combo +s"want^to" sample.cha
combo +s"want^to" +d sample.cha
combo +s"want^to" +d1 sample.cha | freq
combo +d2 +s"_*^." sample.cha | freq
```

This final command provides a useful way of searching for utterance final words and tabulating their frequency. The use of the +d4 switch was described in the previous section.

+g COMBO can operate in either string-oriented or word-oriented mode. The default mode is word-oriented. COMBO can be converted to a string-oriented program by using the +g option. Word-oriented search assumes that the string of characters requested in the search string is surrounded by spaces or other word delimiting characters. The string-oriented search does not make this assumption. It sees a string of characters simply as a string of characters. In most cases, there is no need to use this switch, because the default word-oriented mode is usually more useful.

The interpretation of metacharacters varies depending on the search mode. In word-oriented mode, an expression with the asterisk metacharacter, such as **air*^plane**, will match *air plane* as well as *airpline plane* or *airy plane*. It will not match *airplane* because, in word-oriented mode, the program expects to find two words. It will not match *air in the plane* because the text is broken into words by assuming that all adjacent nonspace characters are part of the same word, and a space marks the end of that word. You can think of the search string *air* as a signal for the computer to search for the expressions: **_air_**, **_air.**, **air?**, **air!**, and so forth, where the underline indicates a space.

The same expression **air*^plane** in the string-oriented search will match *airline plane, airy plane, air in the plane* or *airplane*. They will all be found because the search string, in this case, specifies the string consisting of the letters "a," "i," and "r", followed by any number of characters, followed by the string "p," "l," "a," "n," and "e." In string-oriented search, the expression (**air^plane**) will match *airplane* but not *air plane* because no space character was specified in the search string. In general, the string-oriented mode is not as useful as the word-oriented mode. One of the few cases when this mode is useful is when you want to find all but some given forms. For example if you are looking for all the forms of the verb *kick* except the *ing* form, you can use the expression "kick*^! ^!ing" and the +g switch.

+o The +t switch is used to control the addition or deletion of particular tiers or lines from the input and the output to COMBO. In some cases, you may want to include a tier in the output that is not being included in the input. This typically happens when you want to match a string in only one dependent tier, such as the %mor tier, but you want all tiers to be included in the output. In order to do this you would use a command of the following shape:

```
combo +t%mor +s"*ALL" +o% sample2.cha
```

+s This option is obligatory for COMBO. It is used to specify a regular expression to search for in a given data line(s). This option should be immediately followed by the regular expression itself. The rules for forming a regular expression are discussed in detail earlier in this section.

+t Particular dependent tiers can be included or excluded by using the +t option immediately followed by the tier code. By default, COMBO excludes the header and dependent code tiers from the search and output. However, when the dependent code tiers are included by using the +t option, they are combined with their speaker tiers into clusters. For example, if the search expression is **the^*^kitten**, the match would be found even if *the* is on the speaker tier and *kitten* is on one of the speaker's associated dependent tiers. This feature is useful if one wants to select for analyses only speaker tiers that contain specific word(s) on the main tier and some specific codes on the dependent code tier. For example, if one wants to produce a frequency count of the words *want* and *to* when either one of them is coded as an imitation on the %spa line, or *neat* when it is a continuation on the %spa line, the following two commands could be used:

```
combo +s(want^to^*^%spa:^*^$INI*)+(neat^*^%spa:^*^$CON*)
       +t%spa +f +d sample.cha
freq +swant +sto +sneat sample.cmb
```

In this example, the +s option specifies that the words *want, to,* and *$INI* may occur in any order on the selected tiers. The +t%spa option must be added in order to allow the program to look at the %spa tier when searching for a match. The +d option is used to specify that the information produced by the program, such as file name, line number and exact position of words on the tier, should be excluded from the output. This way the output is in a legal CHAT format and can be used as an input to another CLAN program, FREQ in this case. The same effect could also be obtained by using the piping feature.

+x COMBO searches are sequential. If you specify the expression **dog^cat**, the program will match only the word “dog” directly followed by the word “cat”. If you want to find clusters that contain both of these words, in any order, you need to use the +x option. This option allows the program to find the expressions in both the original order and in reverse order. Thus, to find a combination of “want” and “to” anywhere and in any order, you use this command:

```
combo +swant^to +x sample.cha
```

COMBO also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.5 COOCUR

The COOCCUR program tabulates co-occurences of words. This is helpful for analyzing syntactic clusters. By default, the cluster length is two words, but you can reset this value just by inserting any integer up to 20 immediately after the +n option. The second word of the initial cluster will become the first word of the following cluster, and so on.

```
cooccur +t*MOT +n3 sample.cha +f
```

The +t*MOT switch tells the program to select only the *MOT main speaker tiers. The header and dependent code tiers are excluded by default. The +n3 option tells the program to combine three words into a word cluster. The program will then go through all of *MOT main speaker tiers in the sample.cha file, three words at a time. When COOCCUR reaches the end of an utterance, it marks the end of a cluster, so that no clusters are broken across speakers or across utterances. Co-ocurrences of codes on the %mor line can be searched using commands such as this example:

```
cooccur +t%mor -t* +s*def sample2.cha
```

8.5.1 Unique Options

+d Strip the numbers from the output data that indicate how often a particular cluster occurred.

+n Set cluster length to a particular number. For example, +n3 will set cluster length to 3.

+s Select either a word or a file of words with @filename to search for.

COOCCUR also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.
have a final delimiter. It can also be used to insert final periods on other lines.

8.6 DIST

This program produces a listing of the average distances between words or codes in a file. DIST computes how many utterances exist between occurrences of a specified key word or code. The following example demonstrates a use of the DIST program.

```
dist +t%spa -t* +b: sample.cha
```

This command line tells the program to look at the %spa tiers in the file sample.cha for codes containing the : symbol. It then does a frequency count of each of these codes, as a group, and counts the number of turns between occurrences. The -t* option causes the program to ignore data from the main speaker tiers.

8.6.1 Unique Options

+b This option allows you to specify a special character after the +b. This character is something like the colon that you have chosen to use to divide some complex code into its component parts. For example, you might designate a word as a noun on the dependent tier then further designate that word as a pronoun by placing a code on the dependent tier such as $NOU:pro. The program would analyze each element of the complex code individually and as a class. For the example cited earlier, the program would show the distance between those items marked with a $NOU (a larger class of words) and show the distance between those items marked with $NOU:pro as a subset of the larger set. The +b option for the example would look like this with a colon following the +b:

```
dist +b: sample.cha
```

+d Output data in a form suitable for statistical analysis.

+g Including this switch in the command line causes the program to count only one occurrence of each word for each utterance. So multiple occurrences of a word or code will count as one occurrence.

+o This option allows you to consider only words that contain the character specified by the b option, rather than all codes in addition to those containing your special character.

DIST also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.7 DSS

This program is designed to provide an automatic computation of the Developmental Sentence Score (DSS) of Lee (1974). This score is based on the assignment of scores for a variety of syntactic, morphological, and lexical structures across eight grammatical domains. The computation of DSS relies on the part of speech (POS) analysis of the %mor tier.

8.7.1 CHAT File Format Requirements

For DSS to run correctly on a file, the following CHAT conventions must be followed:

1. All utterances must have delimiters, and imperatives must end with an exclamation mark.
2. Incomplete or interrupted utterances must end either with the +... or the +/. codes.
3. Only the pronoun "I" and the first letter of proper nouns should be in uppercase.
4. Utterances that contain a noun and a verb in a subject-predicate relation in an unusual word order must contain a [+ dss] postcode after the utterance delimiter.
5. DSS automatically excludes any child utterances that are imitations of the immediately preceding adult utterance. If, however, the analyst feels that there are additional child utterances that are imitations and should be excluded from the analysis, the [+ imit] postcode must be included for these utterances.

8.7.2 Selection of a 50-sentence Corpus

DSS scores are based on analysis of a corpus of 50 sentences. The dss program is designed to extract a set of 50 sentences from a language sample using Lee's six inclusion criteria.

1. **The corpus should contain 50 complete sentences.** A sentence is considered complete if it has a noun and a verb in the subject-predicate relationship. To check for this, the program looks for a nominal phrase followed by a verb. Imperatives such as "Look!" also are included. Imperative sentences must have end with an exclamation mark. Immature sentences containing word order reversals such as "car a garage come out" or "hit a finger hammer Daddy" also

should be included. However, these sentences must contain the [+ dss] code after the utterance delimiter on the main tier to be included in the analysis.

2. **The speech sample must be a block of consecutive sentences.** To be representative, the sentences constituting the corpora must occur consecutively in a block, ignoring incomplete utterances. The analyst may use his or her discretion as to which block of sentences are the most representative. The DSS program automatically includes the first 50 consecutive sentences in the transcript. If you wish to start the analysis at some other point, you can use the +z switch in combination with KWAL and piping to DSS.
3. **All sentences in the language sample must be different.** Only unique child sentences will be included in the corpora. DSS automatically analyzes each sentence and excludes any repeated sentences.
4. **Unintelligible sentences should be excluded from the corpus.** The DSS program automatically excludes any sentences containing unintelligible segments. Thus, any sentence containing xxx, xx, yyy, and yy codes on the main tier will be excluded from the analysis.
5. **Echoed sentences should be excluded from the corpus**. Any sentence that is a repetition of the adult's preceding sentence is automatically excluded. Additionally, sentences containing a [+ imit] post-code also may be excluded by using the -s option.
6. **Incomplete sentences should be excluded**. Any sentence that has the +... or the +/. sentence delimiters, indicating that they were either incomplete or interrupted, will not be included in the analysis.
7. **DSS analysis can only be used if at least 50% of the utterances are complete sentences as defined by Lee**. If fewer than 50% of the sentences are complete sentences, then the Developmental Sentence Type analysis (DST) is appropriate instead.

8.7.3 Automatic Calculation of DSS

In order to compute DSS, the user must first complete a morphological analysis of the file using the MOR program with the +c option. After completing the MOR analysis, the %mor line should be disambiguated using POST. Once the disambiguated %mor is created, the user can run DSS to compute the Developmental Sentence Analysis. The DSS program has two modes: automatic and interactive. The use of the **+e** option invokes the automatic mode. A basic automatic DSS command has this shape:

```
dss +b*CHI +le +e sample.mor
```

8.7.4 Interactive Calculation

In the interactive mode, DSS analyzes each sentence in the corpora and then allows the user to add additional sentence points or attempt marks where appropriate. An additional sentence point is assigned to each sentence if it "meets all the adult standard rules" (Lee, p. 137). Sentence points should be withheld for all errors, including those outside the eight categories analyzed by DSS, such as errors in the use of articles, prepositions, plural and possessive markers, and word order changes. In addition, sentence points should be withheld for semantic errors including neologisms such as "sitting-thing" for "chair" or "letterman" for "mailman" (Lee, p. 137).

Grammatical category points should be assigned only to those structures that meet all of Lee's requirements. If a grammatical structure is attempted but produced incorrectly then attempt marks should be inserted in the place of a numerical score. When using the interactive mode, the DSS program displays each sentence and asks the user to determine if it should or should not receive the additional sentence point and allows the user the opportunity to add attempt marks or edit the scoring. When assigning the sentence point, the user can assign a point by typing *p*, can assign no sentence point by typing *n*, or can modify the point values for each of the categories by typing *e* and then typing *p* or *n*.

It is also possible to modify the points given for each category. Here is an example of a display of category points:

```
       Sentence          |IP |PP |PV |SV |NG |CNJ|IR |WHQ|
    what this say.       | 1 |   |   |   |   |   |   | 2 |
```

To edit this display, you should type the name of the column and a plus or minus with a number for how you want the score raised or lowered. For example, if you wish to raise the IP column by 2 points, you type: ip+2. Adding attempt marks is done in a similar fashion. To add the "-" attempt mark to primary verbs you type: pv+0. To remove the attempt marker (-) from primary verbs, you type pv-0.

For example, in the sentence "what this say" the user might want to add attempt markers to both the primary verb (PV) and the interrogative reversal (IR) categories indicating the nature of the grammatical errors. To add an attempt mark for the primary verb category, the user would type: pv+0 and get the following changes:

```
       Sentence          |IP |PP |PV |SV |NG |CNJ|IR |WHQ|
    what this say.       | 1 |   | - |   |   |   |   | 2 |
```

To add an attempt mark for the interrogative reversal category the user would type **ir+0**, which would produce:

```
       Sentence          |IP |PP |PV |SV |NG |CNJ|IR |WHQ|
    what this say.       | 1 |   | - |   |   |   | - | 2 |
```

The DSS program allows the user to make multiple changes simultaneously. There should be no spaces between the *ir* the + and the *0*. This interactive component also enables users to add or subtract point values from grammatical categories in the same way as adding or removing attempt marks.

Warning: The automatic form of DSS is unable to correctly assign points for the following three forms. If these forms are present, they have to be scored using interactive DSS after use of automatic DSS.

1. The pronominal use of "one" as in "One should not worry about one's life." These constructions should receive 7 points as personal pronouns.
2. The distinction between non-complementing infinitive structures (e.g,. I stopped to play) which receives 3 points as secondary verb and later infinitival complement structures (e.g., I had to go), which receive 5 points as secondary verbs.

When these constructions occur in the analysis the DSS program presents both the 3 and the 5 point value, and the user needs to differentiate these.

3. Wh-questions with embedded clauses that do not contain a conjunction (e.g., Why did the man we saw yesterday call you?) in contrast to those where the embedded clause is marked with a conjunction (e.g., What did the man *that* we saw yesterday say to you?).

8.7.5 DSS O**utput**

Once all 50 sentences have been assigned sentence points, the DSS program automatically generates a table. For both the automatic and interactive modes, each sentence is displayed on the left hand column of the table with the corresponding point values. For the interactive mode, the attempt markers for each grammatical category, sentence point assignments, and the DSS score also are displayed. The Developmental Sentence Score is calculated by dividing the sum of the total values for each sentence by the number of sentences in the analysis.

The output of the table has specifically been designed for users to determine "at a glance" areas of strength and weakness for the individual child for these eight grammatical categories. The low points values for both the indefinite and personal pronoun (IP, PP) categories in the table below indicate that this child used earlier developing forms exclusively. In addition, the attempt mark for the main verb (MV) and interrogative reversal (IR) categories suggest possible difficulties in question formulation.

```
Sentence            |IP |PP |MV |SV |NG |CNJ|IR |WHQ|S|TOT|
I like this.        | 1 | 1 | 1 |   |   |   |   |   |1|  4|
I like that.        | 1 | 1 | 1 |   |   |   |   |   |1|  4|
I want hot dog.     |   | 1 | 1 |   |   |   |   |   |0|  2|
I like it .         | 1 | 1 | 1 |   |   |   |   |   |1|  4|
what this say.      | 1 |   | - |   |   |   | - | 2 |0|  3|
```

Developmental Sentence Score: 4.2

8.7.6 DSS **Summary**

DSS has been designed to adhere as strictly as possible to the criteria for both sentence selection and scoring outlined by Lee. The goal is the calculation of DSS scores based upon Lee's (1974) criteria, as outlined below. The numbers indicate the scores assigned for each type of usage.

Indefinite Pronouns (IP) (A)

1	it, this, that
2	no, some, more, all, lot(s), one(s), two (etc.), other(s), another
3	something, somebody, someone
4	nothing, nobody, none, no one
5	any, anything, anybody, anyone, every, everyone, everything, everybody
6	both, few, many, each, several, most, least, last, second, third (etc.)

Personal Pronouns (PP) (B)

1 1st and 2nd person: I, me, my, mine, your(s)
2 3rd person: he, him, his, she, her(s)
3 plurals: we, us, our(s) they, them, their
4 these, those
5 reflexives: myself, yourself, himself, herself, itself, themselves, ourselves
6 Wh-pronouns: who, which, whose, whom, what, how much
Wh-word + infinitive: I know *what* to do, I know *who(m)* to take.
7 (his) own, one, oneself, whichever, whoever, whatever
Each has his own. Take whatever you like.

Main Verb (MV) (C)
1 uninflected verb
copula, is or 's. It's red.
2 is + verb + ing
3 -s and -ed
irregular past, *ate, saw*
copula *am, are, was, were*
auxiliary *am, are, was, were*
4 can, will may + verb
obligatory do + verb
emphatic do + verb
5 could, would, should, might + verb
obligatory does, did + verb
emphatic does, did +verb
6 must, shall + verb
have + verb + en
have got
7 passive including with *get* and *be*
8 have been + verb + ing, had been + verb + ing
modal + have + verb + en
modal + be + verb + ing
other auxiliary combinations (e.g., should have been sleeping)

Secondary Verbs (SV) (D)
1 five early developing infinitives
I wanna see, I'm gonna see, I gotta see, Lemme see, Let's play
2 noncomplementing infinitives: I stopped *to play*
3 participle, present or past: I see a boy *running*. I found the vase *broken*.
4 early infinitives with differing subjects in basic sentences:
I want you *to come*
later infinitival complements: I had *to go*
obligatory deletions: Make it [*to*] go
infinitive with wh-word: I know what *to get*
5 passive infinitive with *get:* I have *to get dressed*
with *be*: I want *to be pulled.*
8 gerund: *Swinging* is fun.

Negative (NG) (E)
1 it, this, that + copula or auxiliary is, ’s + not: It’s not mine.
This is not a dog.
2 can’t don’t
3 isn’t won’t
4 any other aux-negative contractions: *aren’t, couldn’t*
any other pro-aux + neg forms: *you’re not, he’s not*
5 uncontracted negatives with have: I have not eaten it.

Conjunction (CNJ) (F)
1 and
2 but
3 because
4 so, and so, so that, if
5 or, except, only
6 where, when, how, while, whether, (or not), till, until, unless, since,

before, after, for, as, as + adjective + as, as if, like, that, than
wh-words + infinitive: I know *how* to do it.
7 therefore, however, whenever, wherever, etc.

Interrogative Reversal (IR) (G)
1 reversal of copula: *is*n’t *it* red?
2 reversal of auxiliary be: *Is* he coming?
3 obligatory -do, -does, -did *Do* they run?
reversal of modal: *Can* you play?
tag questions: It’s fun *isn’t* it?
4 reversal of auxiliary have: *Has he* seen you?
reversal with two auxiliaries: *Has he been* eating?
5 reversal with three auxiliaries: *Could he have been going?*

Wh-question (WHQ) (H)
1 who, what, what + noun
2 where, how many, how much, what....do, what....for
3 when, how, how + adjective
4 why, what it, how come, how about + gerund
5 whose, which, which + noun

8.7.7 DSS for Japanese

DSS can be used for Japanese data to provide an automatic computation of the Developmental Sentence Score for Japanese (DSSJ; Miyata, & al. 2009) based on the Developmental Sentence Score of Lee (1974). The DSSJ scores are based on a corpus of 100 utterances with disambiguated %mor tiers. The basic command has this shape:

```
dss +lj +ddssrulesjp.cut +b*CHI +c100 +e *.cha
```

The items scored by DSSJ are listed below. The numbers indicate the scores assigned for each type of usage. The morpological codes refer to the codes used in JMOR04 and WAKACHI2002 v.4.

Verb Final Inflection (Vlast)

1	PAST, PRES, IMP:te
2	HORT, CONN
3	COND:tara
4	CONN&wa, SGER, NEG&IMP:de
5	IMP, NEG&OBL

Verb Middle Inflection (Vmid)

1	COMPL, NEG, sub|i
2	DESID, POT, POL, sub|ku, sub|ik
3	sub|mi, sub|ar, sub|ok, sub|age
4	PASS
5	sub|moraw, sub|kure

Adjective Inflection (ADJ)

1	A-PRES
3	A-NEG-. A-ADV
4	A-PAST

Copula (COP)

1	da&PRES
3	de&wa-NEG-PRES, de&CONN
4	da-PAST, da&PRES:na, ni&ADV
5	de&CONN&wa

Adjectival Nouns + Copula (AN+COP)

4	AN+da&PRES, AN+ni&ADV, AN+da&PRES:na

Conjunctive particles (CONJ ptl)

2	kara=causal
3	to, kara=temporal, kedo
4	shi, noni

Conjunctions (CONJ)

4	datte, ja, de/sorede, dakara
5	demo

Elaborated Noun Phrases (NP)

2	N+no+(N), A+N
3	N+to+N, Adn+N, V+N
5	AN+na+N, V+SNR

Compounds (COMP)

1	N+N
5	PROP+N, V+V, N+V

Case Particles (CASE)

1	ga, ni
2	to, de
3	o, kara

5 made
Topic, Focus, Quotation Particles (TOP, FOC, QUOT)
1 wa, mo
2 tte
3 dake, to (quot)
5 kurai, shika
Adverbs (ADV)
2 motto, moo, mata
3 mada, chotto, ippai
4 ichiban, nanka, sugu
5 yappari, sakki
Sentence Final Particles (SFP)
1 yo, no, ne
2 kanaa, mon, ka, naa
3 no+yo
4 yo+ne, kke
Sentence Modalizers (SMOD)
3 desu
4 mitai, deshoo
5 yoo, jan
Formal Nouns (FML)
4 koto
5 hoo

8.7.8 How DSS works

DSS relies on a series of rules stated in the rules.cut file. Each rule matches one of the top level categories in the DSS. Within each rule there is a series of conditions. These conditions have a focus and points. Here is an example:

FOCUS: pro|it+pro:dem|that+pro:dem|this
POINTS: A1

This condition checks for the presence of the pronouns *it, that*, or *this* and assigns one A1 points if they are located. The pattern matching for the Focus uses the syntax of a COMBO search pattern. DSS goes through the sentence one word at a time. For each word, it checks for a match across all of the rules. Within a rule, DSS checks across conditions in order from top to bottom. Once a match is found, it adds the points for that match and then moves on to the next word. This means that, if a condition assigning fewer points could block the application of a condition assigning more points, you need to order the condition assigning more points before the condition assigning fewer points. If there is no blocking relation between conditions, then you do not have to worry about condition ordering. The Japanese implementation of DSS differs from the English implementation in one important way. In Japanese, after a match occurs, no more rules are searched and the processor moves directly on to the next word. In English, on the other hand, after a match occurs, the processor moves on to the next rules before moving on to the next word.

Miyata, S., Hirakawa, M., Ito, K., MacWhinney, B., Oshima-Takane, Y., Otomo, K. Shirai, Y., Sirai, H., & Sugiura, M. (2009). Constructing a New Language Measure for Japanese: Developmental Sentence Scoring for Japanese. In: Miyata, S. (Ed.) *Development of a Developmental Index of Japanese and its application to Speech Developmental Disorders.* Report of the Grant-in Aid for Scientific Research (B) (2006-2008) No. 18330141, Head Investigator: Susanne Miyata, Aichi Shukutoku University. 15-66.

8.7.9 Unique Options

+b Designate which speaker to be analyzed.

+c Determine the number of sentences to be included in analysis. The default for this option is 50 sentences. These sentences must contain both a subject and a verb, be intelligible, and be unique and non-imitative. A strict criteria is used in the development of the corpora. Any sentences containing xxx yyy and www codes will be excluded from the corpora.

+e Automatically generate a DSS table.

+s This switch has specific usage with DSS. To include sentences marked with the [+ dss] code, the following option should be included on the command line: +s"[+ dss]". To exclude sentences with the [+ imit] postcode, the user should include the following option on the command line: -s"[+ imit]". These are the only two uses for the +s/-s option.

Additional options shared across commands can be found in the chapter on Options.

8.8 FREQ

One of the most powerful programs in CLAN is the FREQ program for frequency analysis. It is also one of the easiest programs to use and a good program to start with when learning to use CLAN. FREQ constructs a frequency word count for user-specified files. A frequency word count is the calculation of the number of times a word, as delimited by a punctuation set, occurs in a file or set of files. FREQ produces a list of all the words used in the file, along with their frequency counts, and calculates a type–token ratio. The type–token ratio is found by calculating the total number of unique words used by a selected speaker (or speakers) and dividing that number by the total number of words used by the same speaker(s). It is generally used as a rough measure of lexical diversity. Of course, the type–token ratio can only be used to compare samples of equivalent size, because, as sample size increases, the increase in the number of types starts to level off.

8.8.1 What FREQ Ignores

The CHAT manual specifies two special symbols that are used when transcribing difficult material. The xxx symbol is used to indicate unintelligible speech and the www symbol is used to indicate speech that is untranscribable for technical reasons. FREQ ignores these symbols by default. Also excluded are all the words beginning with one of the following characters: 0, &, +, -, #. If you wish to include them in your analyses, list them, along with other words you are searching for, in a file and use the +s/-s option to specify them on the command line. The FREQ program also ignores header and code tiers by default. Use the +t option if you want to include headers or coding tiers.

8.8.2 Studying Lexical Groups

The easiest way of using FREQ is to ask it to give a complete frequency count of all the words in a transcript. However, FREQ can also be used to study the development and use of particular lexical groups. If you are interested, for example, in how children use personal pronouns between the ages of 2 and 3 years, a frequency count of these forms would be helpful. Other lexical groups that might be interesting to track could be the set of all conjunctions, all prepositions, all morality words, names of foods, and so on. In order to get a listing of the frequencies of such words, you need to put all the words you want to track into a text file, one word on each line by itself, and then use the +s switch with the name of the file preceded by the @ sign, as in this example:

```
freq +s@articles.cut +f sample.cha
```

This command would conduct a frequency analysis on all the articles that you have put in the file called articles.cut. You can create the articles.cut file using either the CLAN editor in Text Mode or some other editor saving in "text only." The file looks just like this:

```
a
the
an
```

8.8.3 Building Concordances with FREQ

CLAN is not designed to build final, publishable concordances. However, you can produce simple concordance-like output using the +d0 switch with FREQ. Here is a fragment of the output from the use of this command. This fragment shows 8 matches for the word "those" and 3 for the word "throw."

```
8 those
        File "0012.cha": line 655.
      *MOT:        look at those [= toys] .
        File "0012.cha": line 931.
      *MOT:        do-'nt you touch those wire-s .
        File "0012.cha": line 1005.
      *MOT:        you can-'nt play in those drawer-s .
        File "0012.cha": line 1115.
      *MOT:        those [= crayons] are (y)icky .
        File "0012.cha": line 1118.
      *MOT:       those [= crayons] are (y)icky .
        File "0012.cha": line 1233.
      *MOT:       you can-'nt eat those [= crayons] .
        File "0012.cha": line 1240.
      *MOT:       no # you can-'nt eat those [= crayons] .
        File "0012.cha": line 1271.
      *MOT:       (be)cause you-'re gonna [: go-ing to] put those [= crayons] in
              your mouth .
3 throw
        File "0012.cha": line 397.
      *MOT:       can you <throw that [= football] ?> [>]
        File "0012.cha": line 702.
      *MOT:       yeah # can we throw it [= ball] ?
        File "0012.cha": line 711.
      *MOT:       can you throw that [= ball] to Mommy ?
```

8.8.4 Using Wildcards with FREQ

Some of the most powerful uses of freq involve the use of wildcards. Wildcards are particularly useful when you want to analyze the frequencies for various codes that you have entered into coding lines. Here is an example of the use of wildcards with codes. One line of Hungarian data in sample2.cha has been coded on the %mor line for syntactic role and part of speech, as described in the CHAT manual. It includes these codes: N:A|duck-ACC, N:I|plane-ACC, N:I|grape-ALL, and N:A|baby-ALL, where the suffixes mark accusative and illative cases and N:A and N:I indicate animate and inanimate nouns. If you want to obtain a frequency count of all the animate nouns (N:A) that occur in this file, use this command line:

```
freq +t%mor +s"N:A|*" sample2.cha
```

The output of this command will be:

```
1 n:a|baby-all
1 n:a|ball-acc
1 n:a|duck-acc
```

Note that material after the +s switch is enclosed in double quotation marks to guarantee that wildcards will be correctly interpreted. For Macintosh and Windows, the double quotes are the best way of guaranteeing that a string is correctly interpreted. On Unix, double quotes can also be used. However, in Unix, single quotes are necessary when the search string contains a $ sign.

The next examples give additional search strings with asterisks and the output they will yield when run on the sample file. Note that what may appear to be a single underline in the second example is actually two underline characters.

String **Output**

```
*-acc               1 n:a|ball-acc
                    1 n:a|duck-acc
                    1 n:i|plane-acc

*-a__               1 n:a|baby-all
                    1 n:a|ball-acc
                    1 n:a|duck-acc
                    1 n:i|grape-all
                    1 n:i|plane-acc

N:*|*-all           1 N:A|baby-all
                    1 N:I|grape-all
```

These examples show the use of the asterisk as a wildcard. When the asterisk is used, FREQ gives a full output of each of the specific code types that match. If you do not want to see the specific instances of the matches, you can use the percentage wildcard, as in the following examples:

```
String                   Output

N:A|%               3 N:A|
%-ACC               3 -ACC
%-A__               3 -ACC
                    2 -ALL
N:%|%-ACC           3 N:|-ACC
N:%|%               5 N:|
```

It is also possible to combine the use of the two types of wildcards, as in these examples:

```
String                   Output

N:%|*-ACC           1 N:|ball-acc
                    1 N:|duck-acc
                    1 N:|plane-acc
N:*|%               3 N:A|
                    2 N:I|
```

Researchers have also made extensive use of FREQ to tabulate speech act and interactional codes. Often such codes are constructed using a taxonomic hierarchy. For example, a code like $NIA:RP:NV has a three-level hierarchy. In the INCA-A system discussed in the chapter on speech act coding in the CHAT manual, the first level codes the interchange type; the second level codes the speech act or illocutionary force type; and the third level codes the nature of the communicative channel. As in the case of the morphological example cited earlier, one could use wildcards in the +s string to analyze at different levels. The following examples show what the different wildcards will produce when analyzing the %spa tier. The basic command here is:

```
freq +s"$*" +t%spa sample.cha
```

String	Output
`$*`	frequencies of all the three-level codes in the %spa tier
`$*:%`	frequencies of the interchange types
`$%:*:%`	frequencies of the speech act codes
`$RES:*: %`	frequencies of speech acts within the RES category
`$*:sel:%`	frequencies of the interchange types that have SEL speech acts

If some of the codes have only two levels rather than the complete set of three levels, you need to use an additional % sign in the +s switch. Thus the switch

```
+s"$%:*:%%"
```

will find all speech act codes, including both those with the third level coded and those with only two levels coded.

As another example of how to use wild cards in FREQ, consider the task of counting all the utterances from the various different speakers in a file. In this case, you count the three-letter header codes at the beginnings of utterances. To do this, you need the +y switch in order to make sure FREQ sees these headers. The command is:

```
freq +y +s”\**:” *.cha
```

8.8.5 FREQ on the %mor line

The previous section illustrates the fact that searches for material on the %mor line can be difficult to compose. To help in this regard, we have composed a variant of FREQ searches that takes advantage of the syntax of the %mor tier. To see the available options here, you can type “freq +s@” on the command line and you will see the following information

```
+s@ Followed by file name or morpho-syntax
Morpho-syntax search pattern
    # prefix marker
    | part-of-speech
    r stem of the word
    - suffix marker
    & nonconcatenated morpheme
    = English translation for the stem
  followed by - and  <--does not apply to contractions (~)
    * find any match
    % erase any match
    word -find "word"

Exception:
    ~ contractions
  followed by - and
    * find ONLY contractions
    + find contractions and other matches
    % erase any match

For example:
    +t%mor -t* +s"@r-*,o-%"
  find all stems and erase all other markers
    +t%mor -t* +s"@r-*,|-adv,o-%"
  find all stems of all "adv" and erase all other markers
    +t%mor -t* +s"@r-be"
  find all forms of "be" verb
    +t%mor -t* +s"@r-*,|-*,--%,o-%"
  find all stems, parts-of-speech followed by any suffix and erase
suffix and other markers
```

8.8.6 Lemmatization

Researchers are often interested in computing frequency profiles that are computed using lemmas or root forms, rather inflected forms. For example, they may want to treat "dropped" as an instance of the use of the lemma "drop." In order to perform these types of computations, the +s switch can be used with the @ symbol to refer to various parts of complex structures in the %mor line. This system recognizes the following structures on the %mor line:

Element	Symbol	Example	Representation	Part
prefix	#	unwinding	un#v|wind-PROG	un#
stem	r	unwinding	un#v|wind-PROG	wind
part of speech	|	unwinding	un#v|wind-PROG	v|
suffix	-	unwinding	un#v|wind-PROG	PROG
fusion	&	unwound	un#v|wind&PAST	PAST
translation	=	gato	n|gato=cat	cat
other	o	-	-	-

To illustrate the use of these symbols, let us look at several possible commands. All of these commands take the form: freq +t%mor -t* filename.cha. However, in addition, they add +s switches as given in the second column. In these commands, the asterisk is used to distinguish across forms in the frequency count and the % sign is used to combine across forms.

Function	String
All stems with their parts of speech, merge the rest	+s@"r+*,|+*,o+%"
Only verbs	+s@"|+v"
All forms of the stem "go"	+s@"r+go"
The different parts of speech of the stem "go"	+s@"r+go,|+*,o+%"
The stem "go" only when it is a verb	+s@"r+go,|+v,o+%"
All stems, merge the rest	+s@"r+*,o+%"

Of these various forms, the last one given above would be the one required for conducting a frequency count based on lemmas or stems alone. Essentially CLAN breaks every element on %mor tier into its individual components and then matches either literal strings or wild cards provided by the user to each component.

8.8.7 Directing the Output of FREQ

When FREQ is run on a single file, output can be directed to an output file by using the +f option:

```
freq +f sample.cha
```

This results in the output being sent to sample.frq.cex. If you wish, you may specify a file extension other than .frq.cex for the output file. For example, to have the output sent to a file with the extension .mot.cex, you would specify:

```
freq +fmot sample.cha
```

Suppose, however, that you are using FREQ to produce output on a group of files rather than on a single file. The following command will produce a separate output file for each .cha file in the current directory:

```
freq +f *.cha
```

To specify that the frequency analysis for each of these files be computed separately but stored in a single file, you must use the redirect symbol (>) and specify the name of the output file. For example:

```
freq *.cha > freq.all
```

This command will maintain the separate frequency analyses for each file separately and store them all in a single file called freq.all. If there is already material in the freq.all file, you may want to append the new material to the end of the old material. In this case, you should use the form:

```
freq *.cha >> freq.all
```

Sometimes, however, researchers want to treat a whole group of files as a single database. To derive a single frequency count for all the .cha files, you need to use the +u option:

```
freq +u *.cha
```

Again, you may use the redirect feature to specify the name of the output file, as in the following:

```
freq +u *.cha > freq.all
```

8.8.8 Limiting in FREQ

An important analytic technique available in clan is the process of “limiting” which allows you to focus your analysis on the part of your data files that is relevant by excluding all other sections. Limiting is based on use of the +s, +t, and +z switches. Limiting is available in most of the clan string search programs, but cannot be done within special purpose programs such as chstrinG or check.

1. **Limiting by including or excluding dependent tiers.** Limiting can be used to select out particular dependent tiers. By using the +t and -t options, you can choose to include certain dependent tiers and ignore others. For example, if you select a particular main speaker tier, you will be able to choose the dependent tiers of only that particular speaker. Each type of tier has to be specifically se-

lected by the user, otherwise the programs follow their default conditions for selecting tiers.

2. **Limiting by including or excluding main tiers.** When the -t* option is combined with a switch like +t*MOT, limiting first narrows the search to the utterances by MOT and then further excludes the main lines spoken by MOT. This switch functions in a different way from -t*CHI, which will simply exclude all of the utterances of CHI and the associated dependent tiers.
3. **Limiting by including or excluding sequential regions of lines or words.** The next level of limiting is performed when the +z option is used. At this level only the specified data region is chosen out of all the selected tiers.
4. **Limiting by string inclusion and exclusion.** The +s/-s options limit the data that is passed on to subsequent programs.

Here is an example of the combined use of the first four limiting techniques. There are two speakers, *CHI and *MOT, in sample.cha. Suppose you want to create a frequency count of all variations of the $ini codes found on the %spa dependent tiers of *CHI only in the first 20 utterances. This analysis is accomplished by using this command:

```
freq +t*CHI +t%spa +s"$INI*" -t* +z20u sample.cha
```

The +t*CHI switch tells the program to select the main and dependent tiers associated only with the speaker *CHI. The +t%spa tells the program to further narrow the selection. It limits the analysis to the %spa dependent tiers and the *CHI main speaker tiers. The -t* option signals the program to eliminate data found on the main speaker tier for NIC from the analysis. The +s option tells the program to eliminate all the words that do not match the $INI* string from the analysis. Quotes are needed for this particular +s switch in order to guarantee correct interpretation of the asterisk. In general, it is safest to always use pairs of double quotes with the +s switch. The +z20u option tells the program to look at only the first 20 utterances. Now the FREQ program can perform the desired analysis. This command line will send the output to the screen only. You must use the +f option if you want it sent to a file. By default, the header tiers are excluded from the analysis.

The +/-s switch can also be used in combination with special codes to pick out sections of material in code-switching. For example, stretches of German language can be marked inside a transcript of mostly English productions with this form:

```
*CHI:	<ich meine> [@g] cow drinking.
```

Then the command to ignore German material would be:

```
freq –s”<@g>” *.cha
```

8.8.9 TTR for Lemmas

If you run FREQ on the data on the main speaker tier, you will get a type-token ratio that is grounded on whole word forms, rather than lemmas. For example, “run,” “runs,” and “running” will all be treated as separate types. If you want to treat all forms of the lemma “run” as a single type, you should run the file through MOR and POST to get a disambiguated %mor line. Then you can run FREQ in a form such as this to get a lemma-based TTR.

```
freq -t* +t%mor +s"*\|*-%" +s"*\|*" sample.mor.pst
```

Depending on the shape of your morphological forms, you may need to add some additional +s switches to this sample command.

8.8.10 Studying Unique Words and Shared Words

With a few simple manipulations, FREQ can be used to study the extent to which words are shared between the parents and the child. For example, we may be interested in understanding the nature of words that are used by the child and not used by the mother as a way of understanding the ways in which the child’s social and conceptual world is structured by forces outside of the immediate family. In order to isolate shared and unique words, you can go through three steps. To illustrate these steps, we will use the sample.cha file.

1. Run freq on the child’s and the mother’s utterances using these two commands:

```
freq +d1 +t*MOT +f sample.cha
freq +d1 +t*CHI +f sample.cha
```

The first command will produce a sample.frq.cex file with the mother’s words and the second will produce a sample.fr0.cex file with the child’s words.

2. Next you should run freq on the output files:

```
freq +y +o +u sample.f*
```

The output of these commands is a list of words with frequencies that are either 1 or 2. All words with frequencies of 2 are shared between the two files and all words with frequencies of 1 are unique to either the mother or the child.

3. In order to determine whether a word is unique to the child or the mother, you can run the previous command through a second filter that uses the COMBO program. All words with frequencies of 2 are unique to the mother. The words with frequencies of 1 are unique to the child. Commands that automate this procedure are:

```
freq  +y  +o  +u  sample.f*  |  combo  +y  +s"2"  +d  |  freq  +y  +d1  >
shared.frq

freq +y +o +u *.frq
```

The first command has three parts. The first FREQ segment tags all shared words as having a frequency of 2 and all non-shared words as having a frequency of 1. The COMBO segment extracts the shared words. The second FREQ segment strips off the numbers and writes to a file. Then you compare this file with your other files from the mother using a variant of the command given in the second step. In the output from this final command, words with a frequency of 2 are shared and words with a frequency of 1 are unique to the mother. A parallel analysis can be conducted to determine the words unique to the child. This same procedure can be run on collections of files in which both speakers participate, as long as the speaker ID codes are consistent.

8.8.11 Unique Options

+c Find capitalized words only.

+d Perform a particular level of data analysis. By default the output consists of all selected words found in the input data file(s) and their corresponding frequencies. The +d option can be used to change the output format. Try these commands:

```
freq sample.cha +d0
freq sample.cha +d1
freq sample.cha +d2 +tCHI
```

Each of these three commands produces a different output.

+d0 When the +d0 option is used, the output provides a concordance with the frequencies of each word, the files and line numbers where each word, and the text in the line that matches.

+d1 This option outputs each of the words found in the input data file(s) one word per line with no further information about frequency. Later this output could be used as a word list file for KWAL or COMBO programs to locate the context in which those words or codes are used.

+d2 With this option, the output is sent to a file in a very specific form that is useful for input to STATFREQ. This option also creates a stat.out file to keep track of multiple .frq.cex output files. You do not need to use the +f option with +d2, because this is assumed. Note that you must include a +t specification in order to tell the +d2 option which speaker to track for the STATFREQ analysis. You may also wish to use the +t@ID= method to select your target speakers, as in this example for the ne20 sample files from the New England corpus.

```
freq +d2 +t@ID=”*|Target_Child|*” *.cha
```

+d3 This output is essentially the same as that for +d2, but with only the statistics on types, tokens, and the type–token ratio. This option also creates a “stat.out” file to keep track of multiple .frq.cex output files. Word frequencies are not placed into the output. You do not need to use the +f option with +d3, since this is assumed.

+d4 This switch allows you to output just the type–token information.

+o Normally, the output from FREQ is sorted alphabetically. This option can be used to sort the output in descending frequency. The +o1 level will sort to create a reverse concordance.

FREQ also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.9 FREQMERG

If you have collected a large number of freq output files and you want to merge these counts together, you can use freqmerg to combine the outputs of several runs of the freq program. For example, you could run this command:

```
freq sample*.cha +f
```

This would create sample.frq.cex and sample2.frq.cex. Then you could merge these two counts using this command:

```
freqmerg *.frq.cex
```

The only option that is unique to freqmerg is +o, which allows you to search for a specific word on the main speaker tier. To search for a file that contains a set of words use the form +o@filename.

FREQMERG also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.10 FREQPOS

The freqpos program is a minor variant of freq. What is different about freqpos is the fact that it allows the user to track the frequencies of words in initial, final, and second position in the utterance. This can be useful in studies of early child syntax. For example, using freqpos on the main line, one can track the use of initial pronouns or auxiliaries. For open class items like verbs, one can use freqpos to analyze codes on the %mor line. This would allow one to study, for example, the appearance of verbs in second position, initial position, final position, and other positions.

To illustrate the running of freqpos, let us look at the results of this simple command:

```
freqpos sample.cha
```

Here are the first six lines of the output from this command:

```
1  a                    initial =  0, final =  0, other =  1, one word =  0
1  any                  initial =  0, final =  0, other =  1, one word =  0
1  are                  initial =  0, final =  1, other =  0, one word =  0
3  chalk                initial =  0, final =  3, other =  0, one word =  0
```

```
1  chalk+chalk            initial =  0, final =  1, other =  0, one word =  0
1  delicious              initial =  0, final =  0, other =  1, one word =  0
```

We see here that the word “chalk” appears three times in final position, whereas the word “delicious” appears only once and that is not in either initial or final position. In order to study occurrences in second position, we must use the +d switch as in:

```
freqpos +d sample.cha
```

8.10.1 Unique Options

+d Count words in either first, second, or other positions. The default is to count by first, last, and other positions.

+g Display only selected words in the output. The string following the +g can be either a word or a file name in the @filename notation.

-s The effect of this option for freqpos is different from its effects in the other CLAN programs. Only the negative -s value of this switch applies. The effect of using -s is to exclude certain words as a part of the syntactic context. If you want to match a particular word with freqpos, you should use the +g switch rather than the +s switch.

FREQPOS also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.11 GEM

The gem program is designed to allow you to mark particular parts of a transcript for further analysis. Separate header lines are used to mark the beginning and end of each interesting passage you want included in your gem output. These header tiers may contain “tags” that will affect whether a given section is selected or excluded in the output. If no particular tag information is being coded, you should use the header form @bg with no colon. If you are using tags, you must use the colon, followed by a tab. If you do not follow these rules, check will complain.

8.11.1 Sample Runs

By default, gem looks for the beginning marker @bg without tags and the ending marker @eg, as in this example command:

```
gem sample.cha
```

If you want to be more selective in your retrieval of gems, you need to add code words or tags to both the @bg: and @eg: lines. For example, you might wish to mark all cases of verbal interchange during the activity of reading. To do this, you must place the word “reading” on the @bg: line just before each reading episode, as well as on the @eg: line just after each reading episode. Then you can use the +sreading switch to retrieve only this type of gem, as in this example:

```
gem +sreading sample2.cha
```

Ambiguities can arise when one gem without a tag is nested within another or when two gems without tags overlap. In these cases, the program assumes that the gem being terminated by the @eg line is the one started at the last @bg line. If you have any sort of overlap or embedding of gems, make sure that you use unique tags.

GEM can also be used to retrieve responses to particular questions or particular stimuli used in an elicited production task. The @bg entry for this header can show the number and description of the stimulus. Here is an example of a completed header line:

```
@bg:	Picture 53, truck
```

One can then search for all of the responses to picture 53 by using the +s"53" switch in GEM.

The / symbol can be used on the @bg line to indicate that a stimulus was described out of its order in a test composed of ordered stimuli. Also the & symbol can be used to indicate a second attempt to describe a stimulus, as in 1a& for the second description of stimulus 1a, as in this example:

```
@bg:	1b /
*CHI:	a &b ball.
@bg:	1a /
*CHI:	a dog.
@bg:	1a &
*CHI:	and a big ball.
```

Similar codes can be constructed as needed to describe the construction and ordering of stimuli for particular research projects.

When the user is sure that there is no overlapping or nesting of gems and that the end of one gem is marked by the beginning of the next, there is a simpler way of using GEM, which we call lazy GEM. In this form of GEM, the beginning of each gem is marked by @g: with one or more tags and the +n switch is used. Here is an example:

```
@g:	reading
*CHI:	nice kitty.
@g:	offstage
*CHI:	who that?
@g:	reading
*CHI:	a big ball.
@g:	dinner
```

In this case, one can retrieve all the episodes of “reading” with this command:

```
gem +n +sreading
```

8.11.2 Limiting With GEM

GEM also serves as a tool for limiting analyses. The type of limiting that is done by

GEM is very different from that done by KWAL or COMBO. In a sense, GEM works like the +t switches in these other programs to select particular segments of the file for analysis. When you do this, you will want to use the +d switch, so that the output is in CHAT format. You can then save this as a file or pipe it on to another program, as in this command.:

```
gem +sreading +d sample2.cha | freq
```

Note also that you can use any type of code on the @bg line. For example, you might wish to mark well-formed multi-utterance turns, teaching episodes, failures in communications, or contingent query sequences.

8.11.3 Unique Options

+d The +d0 level of this switch produces simple output that is in legal CHAT format. The +d1 level of this switch adds information to the legal CHAT output regarding file names, line numbers, and @ID codes.

+g If this switch is used, all of the tag words specified with +s switches must appear on the @bg: header line in order to make a match. Without the +g switch, having just one of the +s words present is enough for a match.

```
gem +sreading +sbook +g sample2.cha
```

This will retrieve all of the activities involving reading of books.

+n Use @g: lines as the basis for the search. If these are used, no overlapping or nesting of gems is possible and each @g must have tags. In this case, no @eg is needed, but CHECK and GEM will simply assume that the gem starts at the @g and ends with the next @g.

+s This option is used to select file segments identified by words found on the @bg: tier. Do not use the -s switch. See the example given above for +g. To search for a group of words found in a file, use the form +s@filename.

GEM also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.12 GEMFREQ

This program combines the basic features of FREQ and GEM. Like GEM, it analyzes portions of the transcript that are marked off with @bg and @eg markers. For example, gems can mark off a section of bookreading activity with *@bg: bookreading* and *@eg: bookreading*. Once these markers are entered, you can then run GEMFREQ to retrieve a basic FREQ-type output for each of the various gem types you have marked. For example, you can run this command:

```
gemfreq +sarriving sample2.cha
```

and you would get the following output:

```
GEMFREQ +sarriving sample2.cha
Wed May 12 15:54:35 1999
GEMFREQ (04-May-99) is conducting analyses on:
  ALL speaker tiers
  and ONLY header tiers matching: @BG:; @EG:;
****************************************
From file <sample2.cha>
  2 tiers in gem " arriving":
   1 are
   1 fine
   1 how
   1 you
```

8.12.1 Unique Options

+d The d0 level of this switch produces simple output that is in legal CHAT format. The d1 level of this switch adds information to the legal CHAT output regarding file names, line numbers, and @ID codes.

+g If this switch is used, all of the tag words specified with +s switches must appear on the @bg: header line in order to make a match. Without the +g switch, having just one of the +s words present is enough for a match.

```
gem +sreading +sbook +g sample2.cha
```

This will retrieve all of the activities involving reading of books.

+n Use @g: lines as the basis for the search. If these are used, no overlapping or nesting of gems is possible and each @g must have tags. In this case, no @eg is needed, and both CHECK and GEMFREQ will simply assume that the gem starts at the @g and ends with the next @g.

+o Search for a specific word on the main speaker tier. To search for a file of words use the form +o@filename.

8.13 GEMLIST

The GEMLIST program provides a convenient way of viewing the distribution of gems across a collection of files. For example, if you run GEMLIST on both sample.cha and sample2.cha, you will get this output:

```
From file <sample.cha>
12 @BG
 3 main speaker tiers.
21 @EG
 1 main speaker tiers.
24 @BG
 3 main speaker tiers.
 32 @EG
From file <sample2.cha>
 18 @BG: just arriving
  2 main speaker tiers.
 21 @EG: just arriving
 22 @BG: reading magazines
  2 main speaker tiers.
 25 @EG: reading magazines
 26 @BG: reading a comic book
  2 main speaker tiers.
 29 @EG: reading a comic book
```

GEMLIST can also be used with files that use only the @g lazy gem markers. In that case, the file should use nothing by @g markers and GEMLIST will treat each @g as implicitly providing an @eg for the previous @g. Otherwise, the output is the same as with @bg and @eg markers.

The only option unique to GEMLIST is +d which tells the program to display only the data in the gems. GEMLIST also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.14 KEYMAP

The KEYMAP program is useful for performing simple types of interactional and contingency analyses. KEYMAP requires users to pick specific initiating or beginning codes or “keys” to be tracked on a specific coding tier. If a match of the beginning code or key is found, KEYMAP looks at all the codes on the specified coding tier in the next utterance. This is the “map.” The output reports the numbers of times a given code maps onto a given key for different speakers.

8.14.1 Sample Runs

Here is a file fragment with a set of codes that will be tracked by KEYMAP:

```
*MOT: here you go.
%spa: $INI
*MOT: what do you say?
%spa: $INI
*CHI: thanks.
%spa: $RES
*MOT: you are very welcome.
%spa: $CON
```

If you run the KEYMAP program on this data with the $INI as the +b key symbol, the program will report that $INI is followed once by $INI and once by $RES. The key ($INI in the previous example) and the dependent tier code must be defined for the program. On the coding tier, KEYMAP will look only for symbols beginning with the $ sign. All other strings will be ignored. Keys are defined by using the +b option immediately followed by the symbol you wish to search for. To see how KEYMAP works, try this example:

```
keymap +b$INI* +t%spa sample.cha
```

For Unix, this command would have to be changed to quote metacharacters as follows:

```
keymap +b\$INI\* +t%spa sample.cha
```

KEYMAP produces a table of all the speakers who used one or more of the key symbols, and how many times each symbol was used by each speaker. Each of those speakers is followed by the list of all the speakers who responded to the given initiating speaker, including continuations by the initial speaker, and the list of all the response codes and their frequency count.

8.14.2 Unique Options

+b This is the beginning specification symbol.

+s This option is used to specify the code or codes beginning with the $ sign to treat as possible continuations. For example, in the sample.cha file, you might only want to track $CON:* codes as continuations. In this case, the command would be as follows.

```
keymap +b$* +s"$CON:*" +t%spa sample.cha
```

KEYMAP also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.15 KWAL

The KWAL program outputs utterances that match certain user-specified search words. The program also allows the user to view the context in which any given keyword is used. In order to specify the search words, use the +s option, which allows you to search for either a single word or a whole group of words stored in a file. It is possible to specify as many +s options on the command line as you like.

Like COMBO, the KWAL program works not on lines, but on “clusters.” A cluster is a combination of the main tier and the selected dependent tiers relating to that line. Each cluster is searched independently for the given keyword. The program lists all keywords that are found in a given cluster tier. A simple example of the use of KWAL is:

```
kwal +schalk sample.cha
```

The output of this command tells you the file name and the absolute line number of the cluster containing the key word. It then prints out the matching cluster.

8.15.1 Tier Selection in KWAL

Sometimes you may want to create new files in which some of the tiers in your original files are systematically deleted. For example, you may wish to drop out certain coding tiers that interfere with the readability of your transcript, or you may wish to drop out a tier that will be later recomputed by a program. For example, in order to drop out the %mor tier for all speakers, except CHI, you can use this command:

```
kwal +t*chi +t%mor +o@ +o* -o%mor  +d +f t.cha
```

The two +t switches work as a matched pair to preserve the %mor tier for CHI. The first +o@ switch will preserve the header tiers. The second and third +o switches work as a pair to exclude the %mor lines in the other speakers. However, the -o%mor switch keeps all of the dependent tiers except for %mor. The +t switch is used for selecting parts of the transcript that may also be searched using the +s option. The +o switch, on the other hand, only has an impact on the shape of the output. The +d switch specifies that the output should be in CHAT format and the +f switch sends the output to a file. In this case, there is no need to use the +s switch. Try out variations on this command with the sample files to make sure you understand how it works.

Main lines can be excluded from the analysis using the -t* switch. However, this exclusion affects only the search process, not the form of the output. It will guarantee that no matches are found on the main line, but the main line will be included in the output. If you want to exclude certain main lines from your output, you can use the -o switch, as in:

```
kwal +t*CHI +t%spa -o* sample.cha
```

You can also do limiting and selection by combining FLO and KWAL:

```
kwal +t*CHI +t%spa +s"$*SEL*" -t* sample.cha +d |
flo -t* +t%
```

To search for a keyword on the *MOT main speaker tiers and the %spa dependent tiers of that speaker only, include +t*MOT +t%spa on the command line, as in this command.

```
kwal +s"$INI:*" +t%spa +t*MOT sample.cha
```

If you wish to study only material in repetitions, you can use KWAL in this form:

```
kwal +s”+[//]” *.cha +d3 +d
```

8.15.2 Unique Options

+a Sort the output alphabetically. Choosing this option can slow down processing significantly.

+d Normally, KWAL outputs the location of the tier where the match occurs. When the +d switch is turned on you can output each matched sentence without line number information in a simple legal CHAT format. The +d1 switch outputs legal CHAT format along with file names and line numbers. Try these commands:

```
kwal +s"chalk" sample.cha
kwal +s"chalk" +d sample.cha
kwal +s"chalk" +d1 sample.cha
```

The +d and +d1 switches can be extremely important tools for performing analyses on particular subsets of a text. For example, in one project, a central research question focused on variations in MLU as a function of the nature of the addressee. In order to analyze this, each utterance was given a %add line along with a code that indicated the identity of the addressee. Using sample.cha as an example, the following KWAL line was used:

```
kwal +t%add +t*CHI +s"mot" +d sample.cha | mlu
```

This produced an MLU analysis on only those child utterances that are directed to the mother as addressee.

+/-nS Include or exclude all utterances from speaker S when they occur immediately after a match of a specified +s search string. For example, if you want to exclude all child utterances that follow questions, you can use this command

```
kwal +t*CHI +s"?" -nCHI *.cha
```

+o The +t switch is used to control the addition or deletion of particular tiers or lines from the input and the output to KWAL. In some cases, you may want to include a tier in the output that is not being included in the input. This typically happens when you want to match a string in only one dependent tier, such as the %mor tier, but you want all tiers to be included in the output. In order to do this you would use a command of the following shape:

```
kwal +t%mor +s"*ACC" +o% sample2.cha
```

In yet another type of situation, you may want to include tiers in the KWAL output that are not normally included. For example, if you want to see output with the ages of the children in a group of files you can use this command:

```
kwal +o@Age -t* *.cha
```

+w It is possible to instruct the program to enlarge the context in which the keyword was found. The +w and -w options let you specify how many clusters after and before the target cluster are to be included in the output. These options must be immediately followed by a number. Consider this example:

```
kwal +schalk +w3 -w3 sample.cha
```

When the keyword *chalk* is found, the cluster containing the keyword and the three clusters above (-w3) and below (+w3) will be shown in the output.

KWAL also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.16 MAXWD

This program locates, measures, and prints either the longest word or the longest utterance in a file. It can also be used to locate all the utterances that have a certain number of words or greater.

When searching for the longest word, the MAXWD output consists of: the word, its length in characters, the line number on which it was found, and the name of the file where it was found. When searching for the longest utterance with the +g option, the output consists of: the utterance itself, the total length of the utterance, the line number on which the utterance begins, and the file name where it was found. By default, MAXWD only analyzes data found on the main speaker tiers. The +t option allows for the data found on the header and dependent tiers to be analyzed as well. The following command will locate the longest word in sample.cha.

```
maxwd sample.cha
```

You can also use MAXWD to track all of the words or utterances of a certain length. For example, the following command will locate all of the utterances with only one word in them:

```
maxwd -x1 +g2 sample.cha
```

Alternatively, you may want to use MAXWD to filter out all utterances below or above a certain length. For example, you can use this command to output only sentences with four or more words in them:

```
maxwd +x4 +g2 +d1 +o%
```

8.16.1 Unique Options

+b You can use this switch to either include or exclude particular morpheme delimiters. By default the morpheme delimiters #, ~, and - are understood to delimit separate morphemes. You can force MAXWD to ignore all three of these by using the -b#-~ form of this switch. You can use the +b switch to add additional delimiters to the list.

+c This option is used to produce a given number of longest items. The following command will print the seven longest words in sample.cha.

```
maxwd +c7 sample.cha
```

If you want to print out all the utterances above a certain length, you can use this KWAL command

```
kwal +x4w sample.cha
```

+d The +d level of this switch produces output with one line for the length level and the next line for the word. The +d1 level produces output with only the longest words, one per line, in order, and in legal CHAT format.

+g This switch forces MAXWD to compute not word lengths but utterance lengths. It singles out the sentence that has the largest number of words or morphemes and prints that in the output. The way of computing the length of the utterance is determined by the number following the +g option. If the number is 1 then the length is in number of morphemes per utterance. If the number is 2 then the length is in number of words per utterance. And if the number is 3 then the length is in the number of characters per utterance. For example, if you want to compute the MLU and MLT of five longest utterances in words of the *MOT, you would use the following command:

```
maxwd +g2 +c5 +d1 +t*MOT +o%mor sample.cha | mlu
```

The +g2 option specifies that the utterance length will be counted in terms of numbers of words. The +c5 option specifies that only the five longest utterances should be sent to the output. The +d1 option specifies that individual words, one per line, should be sent to the output. The +o%mor includes data from the %mor line in the output sent to MLU. The | symbol sends the output to analysis by MLU.

+j If you have elected to use the +c switch, you can use the +j switch to further fine-tune the output so that only one instance of each length type is included. Here is a sample command:

```
maxwd +c8 +j +xw sample.cha
```

+o The +o switch is used to force the inclusion of a tier in the output. In order to do this you would use a command of the following shape:

```
maxwd +c2 +j +o%mor sample2.cha
```

+x This option allows you to start the search for the longest item of a certain type at a certain item length. As a result, all the utterances or words shorter than a specified number will not be included in a search. The number specifying the length should immediately follow the +x option. After that, you need to put either “w” for words, “c” for characters, or “m” for morphemes. Try this command:

```
maxwd sample.cha +x6w
```

MAXWD also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on.

8.17 MLT

The MLT program computes the mean number of utterances in a turn, the mean number of words per utterance, and the mean number of words per turn. A turn is defined as a sequence of utterances spoken by a single speaker. Overlaps are not taken into account in this computation. Instead, the program simply looks for sequences of repeated speaker ID codes at the beginning of the main line. These computations are provided for each speaker separately. Note that none of these ratios involve morphemes. If you want to analyze morphemes per utterances, you should use the MLU program.

8.17.1 MLT Defaults

The exact nature of the MLT calculation depends both on what the program includes and what it excludes. The default principles that it uses are as follows:

1. MLT excludes material in angle brackets followed by either [/] or [//]. This can be changed by adding any of these switches:

   ```
   +s+"</>" +s+"<//>"
   ```

2. In order to exclude utterances with a specific postcode, such as [+ bch], you can use the -s switch:

   ```
   -s"[+ bch]"
   ```

   Similarly, you can use +s to include lines that would otherwise be excluded. For example, you may want to use +s”[+ trn]” to force inclusion of lines marked with [+ trn].

3. The following strings are also excluded:

```
www 0* &* +* -* #* $*.
```

Here the asterisk indicates any material following the first symbol until a delimiter. Unlike the MLU program, MLT does not exclude utterances with *xxx* and *yyy* by default.

4. The program considers the following symbols to be word delimiters:

```
. ? ! , ; [ ] < >
```

The space is also a word delimiter.

5. The program considers the following three symbols to be utterance delimiters:

   . ! ?

   as well as the various complex symbols such as +..., which end with one of these three marks.

6. The special symbols xxx and yyy are not excluded from the data. Thus if the utterance consists of those symbols only it will still be counted.

1. Utterances with no speech on the main line can be counted as turns if you add the [+ trn] code, as in this example:

```
CHI:     0. [+ trn]
%spa:    gestures to mother
```

In order to count this utterance as a turn, you can use this switch:

```
+s+"[+ trn]"
```

The second + after the *s* is used to mark the inclusion of something that is usually excluded. This method for including nonverbal activities in mlt was developed by Pan (1994).

8.17.2 Breaking Up Turns

Sometimes speakers will end a turn and no one takes over the floor. After a pause, the initial speaker may then start up a new turn. In order to code this as two turns rather than one, you can insert a "dummy" code for an imaginary speaker called XXX, as in this example from Rivero, Gràcia, and Fernández-Viader (1998):

```
*FAT:   ma::.
%act:   he touches the girl's throat
*FAT:   say mo::m.
@EndTurn
*FAT:   what's that?
%gpx:   he points to a picture that is on the floor
*FAT:   what's that?
```

Using the @EndTurn marker, this sequence would be counted as two turns, rather than as just one.

8.17.3 Sample Runs

The following example demonstrates a common use of the MLT program:

```
mlt sample.cha
```

8.17.4 Unique Options

+cS Look for unit marker S. If you want to count phrases or narrative units instead of sentences, you can add markers such as [c] to make this segmentation of your transcript into additional units. Compare these two commands:

```
mlt sample.cha
mlt +c[c] sample.cha
```

+d You can use this switch, together with the @ID specification described for STATFREQ to produce numbers for a statistical analysis, one per line. The command for the sample file is:

```
mlt +d +t@ID="*|Target_Child*" sample.cha
```

The output of this command would be something like this:

```
eng samp sample 0110 CHI  6  6  8 1.333 1.000 1.333
```

This output gives 11 fields in this order: language, corpus, file, age, participant id, number of utterances, number of turns, number of words, words/turn, utterances/ turn, and words/utterance. The first five of these fields come from the @ID field. The next six are computed for the particular participant for the particular file. In order to run this type of analysis you must have an @ID header for each participant you wish to track. Alternatively, you can use the +t switch in the form +t*CHI. In this case, all of the *CHI lines will be examined in the corpus. However, if you have different names for children across different files, you need to use the @ID fields.

+d1 This level of the +d switch outputs data in another systematic format, with data for each speaker on a single line. However, this form is less adapted to input to a statistical program than the output for the basic +d switch. Also this switch works with the +u switch, whereas the basic +d switch does not. Here is an example of this output:

```
*CHI:  6  6  8 1.333 1.000 1.333
*MOT:  8  7 43 6.143 1.143 5.375
```

+g You can use the +g option to exclude utterances composed entirely of particular words. For example, you might wish to exclude utterances composed only of *hi*, *bye*, or both of these words together. To do this, you should place the words to be excluded in a file, each word on a separate line. The option should be immediately followed by the file name. That is to say, there should not be a space between the +g option and the name of this file. If the file name is omitted, the program displays an error message: "No file name for the +g option specified!"

+s This option is used to specify a word to be used from an input file. This option should be immediately followed by the word itself. In order to search for a group of words stored in a file, use the form +s@filename. The -s value of this switch excludes certain words from the MLT count. This is a reasonable thing to do. The +s switch bases the count only on the included words. It is difficult to imagine why anyone would want to do such an analysis.

MLT also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.18 MLU

The MLU program computes the mean length of utterance, which is the ratio of morphemes to utterances. The predecessor of the current MLU measure was the "mean length of response" or MLR devised by Nice (1925). The MLR corresponds to what we now call MLUw or mean length of utterance in Words. Brown (1973) emphasized the value of thinking of MLU in terms of morphemes, rather than words. Brown was particularly interested in the ways in which the acquisition of grammatical morphemes reflected syntactic growth and he believed that MLUm or mean length of utterance in morphemes would reflect this growth more accurately than MLUw. Brown linked growth in MLU to movement through six stages from MLU 1.75 to MLU 4.5. Subsequent research (Klee, Schaffer, May, Membrino, & Mougey, 1989) showed that MLU is correlated with age until about 48 months. Rondal, Ghiotto, Bredart, and Bachelet (1987) found that MLU is highly correlated with increases in grammatical complexity between MLU of 1 and 3. However, after MLU of 3.0, the measure was not well correlated with syntactic growth, as measured by LARSP. A parallel study by Blake, Quartaro, and Onorati (1970) with a larger subject group found that MLU was correlated with LARSP until MLU 4.5. Even better correlations between MLU and grammatical complexity have been reported when the IPSyn is used to measure grammatical complexity (Scarborough, Rescorla, Tager-Flusberg, Fowler, & Sudhalter, 1991).

Brown (1973, p. 54) presented the following set of rules for the computation (by hand) of MLU:

1. Start with the second page of the transcription unless that page involves a recitation of some kind. In this latter case, start with the first recitation free stretch. Count the first 100 utterances satisfying the following rules.
2. Only fully transcribed utterances are used; none with blanks. Portions of utterances, entered in parentheses to indicate doubtful transcription, are used.
3. Include all exact utterance repetitions (marked with a plus sign in records). Stuttering is marked as repeated efforts at a single word; count the word once in the most complete form produced. In the few cases where a word is produced for emphasis or the like (*no, no, no*) count each occurrence.
4. Do not count such fullers as *mm* or *oh*, but do count *no*, *yeah*, and *hi*.
5. All compound words (two or more free morphemes), propernames, and riualized reduplications coun as single words. Examples: *birthday, rakety-booom, choo-choo, quack-quack, night-night, pocketbook, seesaw*. Justification is that there is no evidence that the constitutent morphemes functions as such for these children.
6. Count as one morpheme all irregular pasts of the verb (*got, did, went, saw*).

Justification is that there is no evidence that the child relates these to present forms.

7. Count as one morpheme all diminutives (*doggie, mommie*) because these children at least to do not seem to use the suffix productively. Diminutives are the standard forms used by the child.
8. Count as separate morphemes all auxiliaries (*is, have, will, can, must, would*). Also all catenatives: *gonna, wanna, hafta*. These latter counted as single morphemes rather than as gong to or want to because evidence is that they function so for the children. Count as separate morphemes all inflections, for example, possessive [s], plural [s], third person singular [s], regular past [d], and progressive [ing].
9. The range count follows the above rules but is always calculated for the total transcription rather than for 100 utterances.

Because researchers often want to continue to follow these rules, it is important to understand how to implement this system in CLAN. Here is a point by point description, corresponding to Brown's nine points.

1. To isolate 100 sentences, use the +z switch. Brown recommended using 100 utterances. He also suggested that these should be taken from the second page of the transcript. In effect, this means that roughly the first 25 utterances should be skipped. The switch that would achieve this effect in the MLU program is +z25u-125u.
2. The MLU programs excludes utterances with unrecognized material transcribed as xxx and xx by default.
3. If you mark repetitions and retraces using the CHAT codes of [/] and [//] the repeated material will be excluded from the computation automatically. This behavior can be changed by using the +r switch in MLU.
4. If you want forms to be treated as nonwords, you can precede them with the marker &, as in &mm. Alternatively, you can add the switch –smm to exclude this form or you can have a list of forms to exclude. The following strings are also excluded by default:

   `xxx yyy www uh um 0* &* +* -* #* $*`

   where the asterisk indicates any material following the exclusion symbol. If *xxx, yyy,* or *www* occur, the whole utterance is skipped. However, the utterance is not skipped for the other symbols, although they are not counted as morphemes. The symbols xx and yy are counted as morphemes. In fact, the symbols *xx* and *yy* are used as variants of *xxx* and *yyy* specifically to avoid exclusion in the MLU program. If the utterance consists of only excludable material, the whole utterance will be ignored. In addition, suffixes, prefixes, or parts of compounds beginning with a zero are automatically excluded and there is no way to modify this exclusion. Brown recommends excluding *mm* and *oh* by default. However, if you want to exclude these filler words, you will need to list them in a file and use the -s switch, as in:

   mlu -s@excludewords sample.cha

   You can use +s to include lines that would otherwise be excluded. For example, you may want to use +s"[+ trn]" to force inclusion of lines marked with [+ trn]. You can use the -sxxx switch to change the exclusionary behavior of MLU. In this case, the program stops excluding sentences that have xxx from the count, but still excludes the specific string “xxx”.

5. When MLU is computed from the %mor line, the compound marker is excluded as a morpheme delimiter, so this restriction is automatic.
6. The & marker for irregular morphology is not treated as a morpheme delimiter, so this restriction is automatic.
7. By default, diminutives are treated as real morphemes. In view of the evidence for the productivity of the diminutive, it is difficult to understand why Brown thought they were not productive.
8. The treatment of hafta as one morpheme is automatic unless the form is replaced by [: have to]. The choice between these codes is left to the transcriber.
9. The range count simply excludes the use of the +z switch.

It is also possible to exclude utterances with a specific postcode, such as [+ bch], using the -s switch:

```
-s"[+ bch]"
```

The use of postcodes needs to be considered carefully. Brown suggested that all sentences with unclear material be excluded. Brown wants exact repetitions to be included and does not exclude imitations. However, other researchers recommend also excluding imitation, self-repetitions, and single-word answers to questions. If you want to have full control over what is excluded by MLU, the best approach is to use a postcode such as [+ exc] for all utterances that you think should be excluded.

The program considers the following three symbols to be morpheme delimiters:

- # ~

MOR analyses distinguish between these delimiters and the ampersand (&) symbol that indicates fusion. As a result, morphemes that are fused with the stem will not be included in the MLU count. If you want to change this list, you should use the +b option described below. For Brown, compounds and irregular forms were monomorphemic. This means that + and & should not be treated as morpheme delimiters for an analysis that follows his guidelines. The program considers the following three symbols to be utterance delimiters:

. ! ?

as well as the various complex symbols such as +... which end with one of these three marks.

The computation of MLU requires you to morphemicize words. The best way to do this is to use the MOR and POST programs to construct a morphemic analysis on the %mor line. This is relatively easy to do for English and other languages for which good MOR grammars and POST disambiguation databases exist. However, if you are working in a language that does not yet have a good MOR grammar, this process would take more time. Even in English, to save time, you may wish to consider using MLU to compute MLUw (mean length of utterance in words), rather than MLU. Malakoff, Mayes, Schottenfeld, and Howell (1999) found that MLU correlates with MLUw at .97 for English. Aguado (1988) found a correlation of .99 for Spanish, and Hickey (1991) found a correlation of .99 for Irish. If you wish to compute MLUw instead of MLU, you can simply refrain from dividing words into morphemes on the main line. If you wish to divide them, you can use the +b switch to tell MLU to ignore your separators.

8.18.1 Including and Excluding in MLU and MLT

Researchers often wish to conduct MLU analyses on particular subsets of their data. This can be done using commands such as:

```
kwal +t*CHI +t%add +s"mot" sample.cha +d | mlu
```

This command looks at only those utterances spoken by the child to the mother as addressee. KWAL outputs these utterances through a pipe to the MLU program. The pipe symbol | is used to indicate this transfer of data from one program to the next. If you want to send the output of the MLU analysis to a file, you can do this with the redirect symbol, as in this version of the command:

```
kwal +t*CHI +t%add +s"mot" sample.cha +d | mlu > file.mlu
```

The inclusion of certain utterance types leads to an underestimate of MLU. However, there is no clear consensus concerning which sentence forms should be included or excluded in an MLU calculation. The MLU program uses postcodes to accommodate differing approaches to MLU calculations. To exclude sentences with postcodes, the -s exclude switch must be used in conjunction with a file of postcodes to be excluded. The exclude file should be a list of the postcodes that you are interested in excluding from the analysis. For example, the sample.cha file is postcoded for the presence of responses to imitations [+ I], yes/ no questions [+ Q], and vocatives [+ V].

For the first MLU pass through the transcript, you can calculate the child’s MLU on the entire transcript by typing:

```
mlu +t*CHI +t%mor sample.cha
```

For the second pass through the transcript you can calculate the child’s MLU according to the criteria of Scarborough (1990). These criteria require excluding the following: routines [+ R], book reading [+ "], fillers [+ F], imitations [+ I], self-repetitions [+ SR], isolated onomatopoeic sounds [+ O], vocalizations [+ V], and partially unintelligible utterances [+ PI]. To accomplish this, an exclude file must be made which contains all of these postcodes. Of course, for the little sample file, there are only a few examples of these coding types. Nonetheless, you can test this analysis using the Scarborough criteria by creating a file called “scmlu” with the relevant codes in angle brackets. Although postcodes are contained in square brackets in CHAT files, they are contained in angle brackets in files used by CLAN. The scmlu file would look something like this:

```
[+ R]
[+ "]
[+ V]
[+ I]
```

Once you have created this file, you then use the following command:

```
mlu +t*CHI -s@scmlu sample.cha
```

For the third pass through the transcript you can calculate the child’s MLU using a still more restrictive set of criteria, also specified in angle brackets in postcodes and in a separate file. This set also excludes one word answers to yes/no questions [$Q] in the file of words to be excluded. You can calculate the child’s MLU using these criteria by typing:

```
mlu +t*CHI -s@resmlu sample.cha
```

In general, exclusion of these various limited types of utterances tends to increase the child's MLU.

8.18.2 Unique Options

+b You can use this switch to either include or exclude particular morpheme delimiters. By default the morpheme delimiters ~, #, and - are understood to delimit separate morphemes. You can force MLU to ignore all three of these by using the -b#-~ switch. You can use the +b switch to add additional delimiters to the list.

+cS Look for unit marker S. If you want to count phrases or narrative units instead of sentences, you can add markers such as [c] to make this segmentation of your transcript into additional units. Compare these two commands:

```
mlu sample.cha
mlu +c[c] sample.cha
```

+d You can use this switch, together with the ID specification described for STATFREQ to produce numbers for a statistical analysis, one per line. The command for the sample file is:

```
mlu +d +tCHI sample.cha
```

The output of this command should be:

```
en|sample|CHI|1;10.4|female|||Target_Child|| 5  7 1.400 0.490
```

This output gives the @ID field, the number of utterances, number of morphemes, morphemes/utterances, and the standard deviation of the MLU. In order to run this type of analysis, you must have an @ID header for each participant you wish to track. You can use the +t switch in the form +tCHI to examine a whole collections of files. In this case, all of the *CHI lines will be examined in the corpus.

+d1 This level of the +d switch outputs data in another systematic format, with data for each speaker on a single line. However, this form is less adapted to input to a statistical program than the output for the basic +d switch. Also this switch works with the +u switch, whereas the basic +d switch do es not. Here is an example of this output:

```
*CHI:  5  7 1.400 0.490
*MOT:  8 47 5.875 2.891
```

+g You can use the +g option to exclude utterances composed entirely of particular words from the MLT analysis. For example, you might wish to exclude utterances composed only of *hi* or *bye.* To do this, you should place the words to be excluded in a file, each word on a separate line. The option should be immediately followed by the file name. That is to say, there should not be a space between the +g option and the name of this file. If the file name is omitted, the program displays an error message: "No file name for the +g option specified!"

+s This option is used to specify a word to be used from an input file. This option should be immediately followed by the word itself. In order to search for a group of words stored in a file, use the form +s@filename. The -s switch excludes certain words from the analysis. This is a reasonable thing to do. The +s switch bases the analysis only on certain words. It is more difficult to see why anyone would want to conduct such an analysis. However, the +s switch also has another use. One can use the +s switch to remove certain strings from automatic exclusion by MLU. The program automatically excludes xxx, 0, uh, and words beginning with & from the MLU count. This can be changed by using this command:

```
mlu +s+uh +s+xxx +s0* +s&* file.cha
```

MLU also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.19 MODREP

The MODREP program matches words on one tier with corresponding words on another tier. It works only on tiers where each word on tier A matches one word on tier B. When such a one-to-one correspondence exists, MODREP will output the frequency of all matches. Consider the following sample file distributed with CLAN as modrep.cha:

```
@Begin
@Participants:	CHI Child
*CHI:	I want more.
%pho:	aI wan mo
%mod:	aI want mor
*CHI:	want more bananas.
%pho:	wa mo nAnA
%mod:	want mor bAn&nAz
*CHI:	want more bananas.
%pho:	wa mo nAnA
%mod:	want mor bAn&nAz
*MOT:	you excluded [//] excluded [/] xxx yyy www
		&d do?
%pho:	yu du
%mod:	yu du
@End
```

You can run the following command on this file to create a model-and-replica analysis for the child’s speech:

```
modrep +b*chi +c%pho +k modrep.cha
```

The output of MODREP in this case should be as follows:

```
From file <modrep.cha>
 1 I
       1 aI
 2 bananas
       2 nAnA
 3 more
       3 mo
 3 want
       1 wan
       2 wa
```

This output tells us that *want* was replicated in two different ways, and that *more* was replicated in only one way twice. Only the child's speech is included in this analysis and the %mod line is ignored. Note that you must include the +k switch in this command in order to guarantee that the analysis of the %pho line is case-sensitive. By default, all CLAN programs are case-insensitive. However, on the %pho line, UNIBET uses capitalization to distinguish between pairs of different phonemes.

8.19.1 Exclusions and Inclusions

By default, MODREP ignores certain strings on the model tier and the main tier. These include xxx, yyy, www, material preceded by an ampersand, and material preceding the retracing markers [/] and [//]. To illustrate these exclusions, try this command:

```
modrep +b* +c%pho +k modrep.cha
```

The output of this command will look like this:

```
MODREP +b* +c%PHO +k modrep.cha
Thu May 13 13:03:26 1999
MODREP (04-May-99) is conducting analyses on:
  ALL speaker main tiers
        and those speakers' ONLY dependent tiers matching: %PHO;
****************************************
From file <modrep.cha>
Model line:
you zzz do ?

is longer than Rep line:
yu du

In File "modrep.cha" in tier cluster around line 13.
```

If you want to include some of the excluded strings, you can add the +q option. For example, you could type:

```
modrep +b* +c%pho +k modrep.cha +qwww
```

However, adding the *www* would destroy the one-to-one match between the model line and the replica line. When this happens, CLAN will complain and then die. Give this a try to see how it works. It is also possible to exclude additional strings using the +q switch. For example, you could exclude all words beginning with "z" using this command:

```
modrep +b* +c%pho +k modrep.cha -qz*
```

However, because there are no words beginning with "z" in the file, this will not change the match between the model and the replica.

If the main line has no speech and only a 0, MODREP will effectively copy this zero as many times as in needed to match up with the number of units on the %mod tier that is being used to match up with the main line.

8.19.2 Using a %mod Line

A more precise way of using MODREP is to construct a %mod line to match the %pho line. In modrep.cha, a %mod line has been included. When this is done the following type of command can be used:

```
modrep +b%mod +c%pho +k modrep.cha
```

This command will compare the %mod and %pho lines for both the mother and the child in the sample file. Note that it is also possible to trace pronunciations of individual target words by using the +o switch as in this command for tracing words beginning with /m/:

```
modrep +b%mod +c%pho +k +om* modrep.cha
```

8.19.3 MODREP and COMBO -- Cross-tier COMBO

MODREP can also be used to match codes on the %mor tier to words on the main line. For example, if you want to find all the words on the main line that match words on the %mor line with an accusative suffix in the mother's speech in sample2.cha, you can use this command:

```
modrep +b%mor +c*MOT +o"*ACC" sample2.cha
```

The output of this command is:

```
From file <sample2.cha>
 1 n:a|ball-acc
      1 labda't
 1 n:a|duck-acc
      1 kacsa't
 1 n:i|plane-acc
      1 repu"lo"ge'pet
```

If you want to conduct an even more careful selection of codes on the %mor line, you can make combined use of MODREP and COMBO. For example, if you want to find all the words matching accusatives that follow verbs, you first select these utterances by running COMBO with the +d switch and the correct +s switch and then pipe the output to the MODREP command we used earlier. This combined use of the two programs can be called "cross-tier COMBO."

```
combo +s"v:*^*^n:*-acc" +t%mor sample2.cha +d |
modrep +b%mor +c*MOT +o"*acc"
```

The output of this program is the same as in the previous example. Of course, in a large input file, the addition of the COMBO filter can make the search much more restrictive and powerful.

8.19.4 Unique Options

+b This switch is used to set the model tier name. There is no default setting. The model tier can also be set to the main line, using +b* or +b*chi.

+c You can use this switch to change the name of the replica tier. There is no default setting.

+n This switch limits the shape of the output from the replica tier in MODREP to some particular string or file of strings. For example, you can cut down the replica tier output to only those strings ending in "-ing." If you want to track a series of strings or words, you can put them in a file and use the @filename form for the switch.

+o This switch limits the shape of the output for the model tier in MODREP to some particular string or file of strings. For example, you can cut down the model tier output to only those strings ending in "-ing" or with accusative suffixes, and so forth. If you want to track a series of strings or words, you can put them in a file and use the @filename form for the switch.

+q The +q switch allows you to include particular symbols such as xxx or &* that are excluded by default. The -q switch allows you to make further exclusions of particular strings. If you want to include or exclude a series of strings or words, you can put them in a file and use the @filename form for the switch.

MODREP also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.20 PHONFREQ

The PHONFREQ program tabulates all of the segments on the %pho line. For example, using PHONFREQ with no further options on modrep.cha will produce this output:

```
 2 A    initial =   0, final =   1, other =   1
 1 I    initial =   0, final =   1, other =   0
 3 a    initial =   1, final =   1, other =   1
 2 m    initial =   2, final =   0, other =   0
 3 n    initial =   1, final =   1, other =   1
 2 o    initial =   0, final =   2, other =   0
 2 w    initial =   2, final =   0, other =   0
```

This output tells you that there were two occurrences of the segment /A/, once in final position and once in other or medial position.

If you create a file called alphabet file and place it in your working directory, you can further specify that certain digraphs should be treated as single segments. This is important if you need to look at diphthongs or other digraphs in UNIBET. In the strings in the alphabet file, the asterisk character can be used to indicate any single character. For example, the string *: would indicate any sound followed by a colon. If you have three instances of a:, three of e:, and three of o:, the output will list each of these three separately, rather than summing them together as nine instances of something followed by a colon. Because the asterisk is not used in either UNIBET or PHONASCII, it should never be necessary to specify a search for a literal asterisk in your alphabet file. A sample alphabet file for English is distributed with CLAN. PHONFREQ will warn you that it does not find an alphabet file. You can ignore this warning if you are convinced that you do not need a special alphabet file.

If you want to construct a complete substitution matrix for phonological analysis, you need to add a %mod line in your transcript to indicate the target phonology. Then you can run PHONFREQ twice, first on the %pho line and then on the %mod line. To run on the %mod line, you need to add the +t%mod switch.

If you want to specify a set of digraphs that should be treated as single phonemes or segments, you can put them in a file called alphabet.cut. Each combination should be entered by itself on a single line. PHONFREQ will look for the alphabet file in either the working directory or the library directory. If it finds no alphabet.cut file, each letter will be treated as a single segment. Within the alphabet file, you can also specify trigraphs that should override particular digraphs. In that case, the longer string that should override the shorter string should occur earlier in the alphabet file.

8.20.1 Unique Options

+b By default, PHONFREQ analyzes the %pho tier. If you want to analyze another tier, you can use the +b switch to specify the desired tier. Remember that you might still need to use the +t switch along with the +b switch as in this command:

```
phonfreq +b* +t*CHI modrep.cha
```

+d If you use this switch, the actual words that were matched will be written to the output. Each occurrence is written out.

+t You should use the +b switch to change the identity of the tier analyzed by PHONFREQ. The +t switch is used to change the identity of the speaker being analyzed. For example, if you want to analyze the main lines for speaker CHI, you would use this command:

```
phonfreq +b* +t*CHI modrep.cha
```

PHONFREQ also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

The lexicon could be much smaller if more rules were written to handle derivational morphology. These would handle prefixes such as “non#” and derivational suffixes such as “-al.” The grammar still needs to be fine-tuned in order to catch common over-regularizations, although it will never be able to capture all possible morphological errors. Furthermore, attempts to capture over regularizations may introduce bogus analyses of good forms, such as “seed” = “*see-PAST.” Other areas for which more rules need to be written include diminutives, and words like “oh+my+goodness,” which should automatically be treated as communicators.

8.21 RELY

This program has two functions. The first is to check reliability. When you are entering a series of codes into files using the Coder Mode, you will often want to compute the reliability of your coding system by having two or more people code a single file or group of files. To do this, you can give each coder the original file, get them to enter a %cod line and then use the RELY program to spot matches and mismatches. For example, you could copy the sample.cha file to the samplea.cha file and change one code in the samplea.cha file. In this example, change the word "in" to "gone" in the code on line 15. Then enter the command

```
rely sample.cha samplea.cha +t%spa
```

The output in the sample.rly file will look like the basic sample.cha file, but with this additional information for line 15:

```
%spa:  $RDE:sel:non $RFI:xxx:gone:?"samplea.cha"
       $RFI:xxx:in:?"sample.cha"
```

If you want the program to ignore any differences in the main line, header line, or other dependent tiers that may have been introduced by the second coder, you can add the +c switch. If you do this, the program will ignore differences and always copy information from the first file. If the command is:

```
rely +c sample.cha samplea.cha +t%spa
```

then the program will use sample.cha as the master file for everything except the information on the %spa tier.

The second function of the RELY program is to allow multiple coders to add a series of dependent tiers to a master file. The main lines of master file should remain unchanged as the coders add additional information on dependent tiers. This function is accessed through the +a switch, which tells the program the name of the code from the secondary file that is to be added to the master file, as in

```
rely +a orig.cha codedfile.cha +t%spa
```

8.21.1 Unique Options

+a Add tiers from second file to first, master, file.

+c Do not check data on nonselected tier.

RELY also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.22 STATFREQ

The STATFREQ program provides a way of producing a summary of word or code frequencies across a set of files. However, within each of the files, you can only look at one speaker at a time. This summary can be sent on as the input to statistical analysis by programs such as Excel, SAS or BMDP. Here is the output from a FREQ run on the *CHI utterances in sample.cha and then a STATFREQ run the output:

```
*CHI: chalk mommy neat that what's yeah Types Token TTR
en|sample|CHI|1;10.04|female|||Target_Child||6 2 1 1 1 1 2 6 8 0.750
```

In order to get this type of output, you need to go through three steps. The actual running of STATFREQ is the last of these three steps.

1. First, you must assign appropriate @ID header lines to the files to be analyzed, using the INSERT program. You can also run INSERT from inside the editor, but starting up CHECK. INSERT will create basic @ID headers which you may wish to modify to add more information. There can be only one @ID header per speaker. These lines take the following shape:

```
@ID:	language|corpus|speaker|age|sex|group|SES|role|situation
```

   an example would be:

```
@ID:	eng|ne20|chi20|1;10.4|m||middle|target_child|situation
```

   Note, that the information here for "group" is missing. This is indicated as missing information between the fifth and sixth bar marks.
2. Next, use FREQ with the +t option followed by the appropriate speaker code. You must also use the +d2 option in the FREQ command line. This will produce a temporary file called stat.out. Here is an example of a FREQ command that outputs data for a STATFREQ analysis:

```
freq +d2 +t@ID=”*|female|*” sample.cha
```

   STATFREQ will produce one line for each file. If your @ID code matches more than one speaker, frequency information from the various speakers that it matches will be merged together. Therefore, you want to make sure that you use the various pieces of information in the @ID field to select out exactly the material you want to match.
3. FREQ will tell you to run STATFREQ by typing:

```
statfreq stat.out.cex +f +d
```

   The result of this command is stat.out.sat.cex. If the @ID header is not found in a given file, the message NO ID GIVEN will be produced by the program.

If you used the +d, then you can import the StatFreq output in stat.out.sat.cex to Excel using this procedure:

Start Excel
select open... from the File menu
choose *.sat.cex as your file
in "Text Import Wizard - Step 1 of 3" select "Delimited" in "Original Data Type"
press "Next >"
in "Text Import Wizard - Step 2 of 3" select "Space" only in "Delimiters"
make sure that "Treat consecutive delimiters as one" is selected
press "Finish"

If you wish to rotate your data table in Excel, here are the commands:

1. Select the data.
2. Right click and copy.
3. Select "Paste Special, Transpose"

The only option unique to STATFREQ is +d which removes the file headers so that the data can be sent directly into a program for statistical analysis. It also replaces missing values with a period, which is usually a symbol representing missing data for statistical analysis. STATFREQ uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.23 TIMEDUR

The TIMEDUR program computes the duration of the pauses between speakers and the duration of overlaps. This program requires a %snd tier created through sonic CHAT. The data is output in a form that is intended for export to a spreadsheet program. Columns labeled with the speaker's ID indicate the length of the utterance. Columns labeled with two speaker ID's, such as FAT-ROS, indicate the length of the pause between the end of the utterance of the first speaker and the beginning of the utterance of the next speaker. Negative values in these columns indicate overlaps.

The only unique option in TIMEDUR is +a, which you can use to specify that the time markers should be taken from the %mov tier instead of the default %snd tier. TIMEDUR also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.24 VOCD

The VOCD command was written by Gerard McKee of the Department of Computer Science, The University of Reading. The research project supporting this work was funded by grants from the Research Endowment Trust Fund of The University of Reading and the Economic and Social Research Council (Grant no R000221995) to D. D. Malvern and B. J. Richards, School of Education, The University of Reading, Bulmershe Court, Reading, England RG6 1HY. An article describing VOCD in detail can be found at http://childes.psy.cmu.edu/manuals/vocd.doc .

Measurements of vocabulary diversity are frequently needed in child language research and other clinical and linguistic fields. In the past, measures were based on the ratio of different words (Types) to the total number of words (Tokens), known as the type–token Ratio (TTR). Unfortunately, such measures, including mathematical transformations of the TTR such as Root TTR, are functions of the number of tokens in the transcript or language sample — samples containing larger numbers of tokens give lower values for TTR and vice versa (Richards & Malvern, 1997a). This problem has distorted research findings (Richards & Malvern, 1997b). Previous attempts to overcome the problem, for example by standardizing the number of tokens to be analyzed from each child, have failed to ensure that measures are comparable across researchers who use different baselines of tokens, and inevitably waste data in reducing analyses to the size of the smallest sample.

The approach taken in the VOCD program is based on an analysis of the probability of new vocabulary being introduced into longer and longer samples of speech or writing. This probability yields a mathematical model of how TTR varies with token size. By comparing the mathematical model with empirical data in a transcript, VOCD provides a new measure of vocabulary diversity called D. The measure has three advantages: it is not a function of the number of words in the sample; it uses all the data available; and it is more informative, because it represents how the TTR varies over a range of token size. The measure is based on the TTR versus token curve calculated from data for the transcript as a whole, rather than a particular TTR value on it.

D has been shown to be superior to previous measures in both avoiding the inherent flaw in raw TTR with varying sample sizes and in discriminating across a wide range of language learners and users (Malvern & Richards, in press; Richards & Malvern, 1998).

8.24.1 Origin of the Measure

TTRs inevitably decline with increasing sample size. Consequently, any single value of TTR lacks reliability as it will depend on the length in words of the language sample used. A graph of TTR against tokens (N) for a transcript will lie in a curve beginning at the point (1,1) and falling with a negative gradient that becomes progressively less steep (see Malvern & Richards, 1997a). All language samples will follow this trend, but transcripts from speakers or writers with high vocabulary diversity will produce curves that lie above those with low diversity. The fact that TTR falls in a predictable way as the token size increases provides the basis for our approach to finding a valid and reliable measure. The method builds on previous theoretical analyses, notably by Brainerd (1982) and in particular Sichel (1986), which model the TTR versus token curve mathematically so that the characteristics of the curve for a transcript yields a valid measure of vocabulary diversity.

Various probabilistic models were developed and investigated in order to arrive at a model containing only one parameter which increases with increasing diversity and falls into a range suitable for discriminating among the range of transcripts found in various language studies. The model chosen is derived from a simplification of Sichel's (1986) type– token characteristic curve and is in the form an equation containing the parameter D. This equation yields a family of curves all of the same general and appropriate shape, with different values for the parameter D distinguishing different members of this family (see Malvern & Richards, 1997). In the model, D itself is used directly as an index of lexical diversity.

In order to calculate D from a transcript, the VOCD program first plots the empirical TTR versus tokens curve for the speaker. It derives each point on the curve from an average of 100 trials on subsamples of words of the token size for that point. The subsamples are made up of words randomly chosen (without replacement) from throughout the transcript. The program then finds the best fit between the theoretical model and the empirical data by a curve-fitting procedure which adjusts the value of the parameter (D) in the equation until a match is obtained between the actual curve for the transcript and the closest member of the family of curves represented by the mathematical model. This value of the parameter for best fit is the index of lexical diversity. High values of D reflect a high level of lexical diversity and lower diversity produces lower values of D.

The validity of D has been the subject of extensive investigation (Malvern & Richards, 1997; Richards & Malvern, 1997a; Richards & Malvern, 1998; Malvern & Richards, in press) on samples of child language, children with SLI, children learning French as a foreign language, adult learners of English as a second language, and academic writing. In these validation trials, the empirical TTR versus token curves for a total of 162 transcripts from five corpora covering ages from 24 months to adult, two languages and a variety of settings, all fitted the model. The model produced consistent values for D which, unlike TTR and even Mean Segmental TTR (MSTTR) (see Richards & Malvern, 1997a: pp. 35-38), correlated well with other well validated measures of language. These five corpora also provide useful indications of the scale for D.

8.24.2 Calculation of D

In calculating D, VOCD uses random sampling of tokens in plotting the curve of TTR against increasing token size for the transcript under investigation. Random sampling has two advantages over sequential sampling. Firstly, it matches the assumptions underlying the probabilistic model. Secondly, it avoids the problem of the curve being distorted by the clustering of the same vocabulary items at particular points in the transcript.

In practice each empirical point on the curve is calculated from averaging the TTRs of 100 trials on subsamples consisting of the number of tokens for that point, drawn at random from throughout the transcripts. This default number was found by experimentation and balanced the wish to have as many trials as possible with the desire for the program to run reasonably quickly. The run time has not been reduced at the expense of reliability, however, as it was found that taking 100 trials for each point on the curve produced consistency in the values output for D without unacceptable delays.

Which part of the curve is used to calculate D is crucial. First, in order to have subsamples to average for the final point on the curve, the final value of N (the number of tokens in a subsample) cannot be as large as the transcript itself. Moreover, transcripts vary hugely in total token count. Second, the equation is an approximation to Sichel's (1986) model and applies with greater accuracy at lower numbers of tokens. In an extensive set of trials, D has been calculated over different parts of the curve to find a portion for which the approximation held good and averaging worked well. As a result of these trials the default is for the curve to be drawn and fitted for N=35 to N=50 tokens in steps of 1 token. Each of these points is calculated from averaging 100 subsamples, each drawn from the whole of the transcript. Although only a relatively small part of the curve is fitted, it uses all the information available in the transcript. This also has the advantage of calculating D from a standard part of the curve for all transcripts regardless of their total size, further providing for reliable comparisons between subjects and between the work of different researchers.

The procedure depends on finding the best fit between the empirical and theoretically derived curves by the least square difference method. Extensive testing confirmed that the best fit procedure was valid and was reliably finding a unique minimum at the least square difference.

As the points on the curve are averages of random samples, a slightly different value of D is to be expected each time the program is run. Tests showed that with the defaults chosen these differences are relatively small, but consistency was improved by VOCD calculating D three times by default and giving the average value as output.

8.24.3 Sample Size

By default, the software plots the TTR versus token curve from 35 tokens to 50 tokens. Each point on the curve is produced by random sampling without replacement. VOCD therefore requires a minimum of 50 tokens to operate. However, the fact that the software will satisfactorily output a value of D from a sample as small as 50 tokens does not guarantee that values obtained from such small samples will be reliable. It should also be noted that random sampling without replacement causes the software to run noticeably more slowly when samples approach this minimum level.

8.24.4 Preparation of Files

Files should be prepared in correct CHAT format and should pass through CHECK, using the +g3 switch to track down spelling and typographical errors. The FREQ program should then be used to create a complete wordlist that can be scanned for further errors. The output from FREQ also allows the researcher to see exactly what FREQ (and therefore VOCD) will treat as a word type. From this information, an exclude file of non-words can be compiled (e.g. hesitations, laughter, etc). These can then be filtered out of the analysis using the -s switch.

8.24.5 The Output from VOCD

To illustrate the functioning of VOCD, let us use a command that examines the child’s output in the file 68.cha in the NE32 sample directory in the lib folder in the CLAN distribution. The +r6 switch here excludes repetitions, the +s@exclude lists a file of words to be excluded, and the +s”*-%%” instructs CLAN to merge across variations of a base word.

```
VOCD +t"*CHI" +r6 -s@exclude +s"*-%%" 68.cha
```

The output of this analysis has four parts:

1. A sequential list of utterances by the speaker selected shows the tokens that will be retained for analysis.
2. A table shows the number of tokens for each point on the curve, average TTR and the standard deviation for each point, and the value of D obtained from the equation for each point. Three such tables appear, one for each time the program takes random samples and carries out the curve-fitting.
3. At the foot of each of the three tables is the average of the Ds obtained from the equation and their standard deviation, the value for D that provided the best fit, and the residuals.
    2. Finally, a results summary repeats the command line and file name and the type and token information for the lexical items retained for analysis, as well as giving the thre optimum values of D and their average.

For the command given above, the last of the three tables and the results summary are:

```
tokens  samples    ttr     st.dev      D
  35      100    0.7963    0.067     54.470
  36      100    0.8067    0.054     60.583
  37      100    0.8008    0.059     59.562
  38      100    0.7947    0.056     58.464
  39      100    0.7831    0.065     55.124
  40      100    0.7772    0.054     54.242
  41      100    0.7720    0.064     53.568
  42      100    0.7767    0.057     56.720
  43      100    0.7695    0.051     55.245
  44      100    0.7650    0.057     54.787
  45      100    0.7636    0.053     55.480
  46      100    0.7626    0.057     56.346
  47      100    0.7543    0.052     54.403
  48      100    0.7608    0.050     58.088
  49      100    0.7433    0.058     52.719
  50      100    0.7396    0.049     52.516


D: average = 55.770; std dev. = 2.289
D_optimum     <55.63; min least sq val = 0.001>



VOCD RESULTS SUMMARY
====================
       Command line:  vocd +t*CHI +r6 -s@exclude +s*-%% 68.cha
          File name:  68.cha
    Types,Tokens,TTR:  <129,376,0.343085>
  D_optimum  values:  <55.36, 55.46, 55.63>
  D_optimum average:  55.48
```

8.24.6 Lemma-based Analysis

If you wish to conduct a lemma-based analysis, rather than a word-based analysis, you can do this from the %mor line using a command of this type:

```
vocd +t%mor –t* +s"*|*-%%" +s"*|*&%%" *.cha
```

8.24.7 Unique Options

+a0 Calculate D_optimum using the split half with evens.

+a1 Calculate D_optimum using the split half with odds.

+c Include capitalized words only.

+d Outputs a list of utterances processed and number of types, tokens and TTR, but does not calculate D.

+d4 Outputs number of types, tokens and TTR only.

+dsN The +ds switch allows separate analysis of odd and even numbered words in the transcript. The results of this can then be fed into a split-half reliability analysis. This switch can have one of two values: +ds0 (for even numbered words) or +ds1 (for odd numbered words).

+**g** Calls up the limiting relative diversity (LRD) sub-routine to compare the relative diversity of two different word classes coded on the %mor tier. This procedure operates in three stages and extracts the words to be included from the %mor tier where the word classes are coded. First, the speaker, the %mor tier and the file name are specified in the usual way, plus the +g switch to invoke the subroutine:

```
vocd +t“*CHI” +t“%mor” -t“*” +g filename
```

Second, the user is prompted twice to specify the word classes to be compared. The following would compare verb and noun diversity and limit the analysis to word stems:

```
+s“v|*” +s“*-%%”

+s“n|*” +s“*-%%”
```

The first word class entered will be the numerator and the second will be the denominator.

+**m** This segments words into their component morphemes so that an overall morpheme diversity measure will be returned. This feature relates particularly to analyses of polysynthetic and agglutinative languages where, because of complex morphological structure, the definition of a ‘word’ may be problematic. In these cases morpheme diversity may be more meaningful the word diversity. VOCD will segment words according to any combination of CHAT morpheme delimiters. This is achieved by placing the boundary markers relevant for the analysis in inverted commas after the -m switch. Thus,

```
vocd +t“*CHI” +m“-#~&+” filename.cha
```

will provide all possible word segmentations.

Hidden Options in *vocd* to Override the Defaults for Curve Fitting and Type of Sampling

```
+Df<n>: D_optimum - size of starting sample (default  35)
+Dt<n>: D_optimum - size of largest sample (default  50)
+Di<n>: D_optimum - size of increments      (default   1)
+Ds<n>: D_optimum - the number of samples   (default 100)
-R    : D_optimum - use random sampling with replacement.
          (default is random sampling without replacement).
```

-E : D_optimum - use sequential sampling.

VOCD also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

8.25 WDLEN

The WDLEN program tabulates word lengths and prints a histogram. The program reads through data files, tabulating the frequencies of various word and utterance lengths. The output consists of word lengths (in characters) and utterance lengths (in words), the frequencies of these lengths, and a histogram of these frequencies. The “Wdlen” in the output represents the word length. The “Utt len” in the output represents the utterance length. THe command allow for a maximum of 100 letters per word and 100 words or morphemes per utterance. If you input exceeds these limits, you will receive an error message. The basic use of the WDLEN program is as follows:

```
wdlen sample.cha
```

The only option unique to WDLEN is +h which allows you to extend the length of the longest line on the histogram. WDLEN also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

9 Options

This chapter describes the various options or switches that are shared across the CLAN analysis commands. To see a list of options for a given program such as KWAL, type *kwal* followed by a carriage return in the Commands window. You will see a list of available options in the CLAN Output window.

Each option begins with a + or a -. There is always a space before the + or -. Multiple options can be used and they can occur in any order. For example, the command:

```
kwal +f +t*MOT sample.cha
```

runs a KWAL analysis on sample.cha. The selection of the +f option sends the output from this analysis into a new file called sample.kwa.cex. The +t*MOT option confines the analysis to only the lines spoken by the mother. The +f and +t switches can be placed in either order.

9.1 +F Option

This option allows you to send output to a file rather than to the screen. By default, nearly all of the programs send the results of the analyses directly to the screen. You can, however, request that your results be inserted into a file. This is accomplished by inserting the +f option into the command line. The advantage of sending the program's results to a file is that you can go over the analysis more carefully, because you have a file to which you can later refer.

The -f switch is used for sending output to the screen. For most programs, -f is the default and you do not need to enter it. You only need to use the -f switch when you want the output to go to the screen for CHSTRING, FLO, and SALTIN. The advantage of sending the analysis to the screen (also called standard output) is that the results are immediate and your directory is less cluttered with nonessential files. This is ideal for quick temporary analysis.

The string specified with the +f option is used to replace the default file name extension assigned to the output file name by each program. For example, the command

```
freq +f sample.cha
```

would create an output file sample.frq.cex. If you want to control the shape of the extension name on the file, you can place up to three letters after the +f switch, as in the command

```
freq +fmot sample.cha
```

which would create an output file sample.mot.cex. If the string argument is longer than three characters, it will be truncated. For example, the command

```
freq +fmother sample.cha
```

would also create an output file sample.mot.cex.

On the Macintosh, you can use the third option under the File menu to set the directory for your output files. On Windows you can achieve the same effect by using the +f switch with an argument, as in:

+fc:	This will send the output files to your working directory on c:.

+f".res"	This sets the extension for your output files.
+f"c:.res"	This sends the output files to c: and assigns the extension .res.

When you are running a command on several files and use the +f switch, the output will go into several files – one for each of the input files. If what you want is a combined analysis that treats all the input files as one large file, then you should use the +u switch. If you want all of the output to go into a single file for which you provide the name, then use the > character at the end of the command along with an additional file name. The > option can not be combined with +f.

9.2 +K Option

This option controls case-sensitivity. A case-sensitive program is one that makes a distinction between uppercase and lowercase letters. The CLAN programs, except for CHSTRING, are not case-sensitive by default. Use of the +k option in all of the other programs overrides the default state and allows them to become case-sensitive as well. For instance, suppose you are searching for the auxiliary verb “may” in a text. If you searched for the word “may” in a case-sensitive program, you would obtain all the occurrences of the word “may” in lower case only. You would not obtain any occurrences of “MAY” or “May.” Searches performed for the word “may” using the +k option produce the words “may,” “MAY,” and “May” as output.

9.3 +P Option

This option allows you to define a custom punctuation set. Because most of the programs in the CLAN system are word-oriented, the beginning and ending boundaries of words must be defined. This is done by defining a punctuation set. The default punctuation set for CLAN includes the space and these characters:

```
, . ; ? ! [ ] < >
```

This punctuation set applies to the main lines and all coding lines with the exception of the %pho and %mod lines which use the UNIBET and PHONASCII systems. Because those systems make use of punctuation markers for special characters, only the space can be used as a delimiter on the %pho and %mod lines.

All of the word-oriented programs have the +p option. This option allows the user to redefine the default punctuation set. This is useful because the CHAT coding conventions use special characters that at times are used as delimiters and other times as parts of words. For example, sometimes the - character is used as a morpheme boundary marker and, therefore, should not be considered part of the word. This is also quite useful when you are working on a language that uses diacritics. To change the punctuation set, you must create a small file that lists all the punctuation marks present in the file. You do this by simply typing out all the punctuation marks on a single line with no spaces between them. This line will change the punctuation set of the main speaker tiers and the code tiers. The name of your new punctuation file should immediately follow the +p in the command line. Here is an example situation. Suppose you wish to change both the main speaker tier and the code tier punctuation sets from the default to the set in newpunct.cut. The contents of the newpunct.cut file are as follows:

```
$*&^!
```

This line indicates the desired punctuation set for the main line and coding tier. You can now issue commands such as the following:

```
freq +pnewpunct.cut sample.cha
```

If you use the +p switch with no file name, the programs look for a file called punct.cut in the current working directory. If you do not use the +p switch at all, the programs look for a punctuation file called punct.cut. If the punct.cut file is not found, the program will then use the default built-in punctuation set. It is advisable to create a punct.cut file when the punctuation characters of the language being analyzed are different from the default punctuation characters. The punct.cut file should contain the new punctuation set and should be located in the current working directory. Because the punct.cut file is referred to automatically, this feature allows you to change the punctuation set once for use with all the CLAN programs. If you do not want CLAN to ever change the default punctuation set, make sure you do not have a punct.cut file in your current working directory and make sure you do not use the +p switch.

9.4 +R Option

This option deals with the treatment of material in parentheses.

+r1 Removing Parentheses. Omitted parts of words can be marked by parentheses, as in "(be)cause" with the first syllable omitted. The +r1 option removes the parentheses and leaves the rest of the word as is.

+r2 Leaving Parentheses. This option leaves the word with parentheses.

+r3 Removing Material in Parentheses. This option removes all of the omitted part.

Here is an example of the use of the first three +r options and their resulting outputs, if the input word is "get(s)":

Option	Output
"no option"	gets
"+r1"	gets
"+r2"	get(s)
"+r3"	get

+r4 Removing Prosodic Symbols in Words. By default, symbols such as #, /, and : are ignored when they occur inside words. Use this switch if you want to include them in your searches. If you do not use this switch, the strings cat and ca:t will be seen as the same. If you use this switch, they will be seen as different. The use of these prosodic marker symbols is discussed in the CHAT manual.

+r5 Text Replacement. By default, material in the form [: text] replaces the material preceding it in the string search programs. If you do not want this replacement, use this switch.

+r6 Retraced Material. By default, material in retracings is included in searches and counts. However, this material can be excluded by using the +r6 switch. In the MLU and MODREP programs, retracings are excluded by default. For these programs, the +r6 switch can be used to include material in retracings.

9.5 +S Option

This option allows you to search for a particular string. The +s option allows you to specify the keyword you desire to find. You do this by putting the word in quotes directly after the +s switch, as in +s"dog" to search for the word "dog." You can also use the +s switch to specify a file containing words to be searched. You do this by putting the file name after the +s preceded by the @ sign, as in +s@adverbs, which will search for the words in a file called adverbs.cut. If you want to look for the literal character @, you need to precede it with a backslash as in +s"\@".

By default, the programs will only search for this string on the main line. Also by default, this switch treats material in square brackets as if it were a single "opaque" form. In effect, unless you include the square brackets in your search string, the search will ignore any material that is enclosed in square brackets. The COMBO program is the only one that allows you to specify regular expressions with this option. The only programs that allow you to include delimiters in the search string are COMBO, FREQ, and KWAL.

It is possible to specify as many +s options on the command line as you like. If you have several +s options specified, the longest ones will be applied first. Use of the +s option will override the default list. For example, the command

```
freq +s"word" data.cut
```

will search through the file data.cut looking for "word."

The +s/-s switch is usually used to include or exclude certain words. However, it can actually be used with five types of material: (1) words, (2) codes or postcodes in square brackets, (3) text in angle brackets associated with particular codes within square brackets, (4) whole utterances associated with particular postcodes, and (5) particular postcodes themselves. Moreover, the switch can be used to either include, exclude, or add particular information. The effect of the switch for the five different types across the three function is described in the following three tables:

Table 4: Search Strings for Inclusion of Five Types of Material

Material	Switch	Results
word	+s"dog"	find only the word "dog"
[code]	+s"[//]"	find only this code
<text>[code]	+s"<//>"	find only text marked by this code
utterance	+s"<+imi>"	find only utterances marked with this postcode
postcode	+s"[+imi]"	find only this postcode itself

Table 5: Search Strings for Exclusion of Four Types of Material

Material	Switch	Results
word	-s"dog"	find all words except the word "dog"
<text>[code]	-s"<//>"	find all text except text marked by this code
utterance	-s"[+ imi]"	find all utterances except utterances marked with this postcode

Table 6: Search Strings for Addition of Four Types of Material that are usually excluded by default

Material	Switch	Results
word	+s+xxx	add "xxx"
[code]	+s+"[//]"	find all text, plus this code
utterance	+s+"<+ bch>"	find all utterances, including those marked with the [+ bch] postcode
postcode	+s+"[+ imi]"	find all text, including this postcode itself

Multiple +s strings are matched as exclusive or's. If a string matches one +s string, it cannot match the other. The most specific matches are processed first. For example, if your command is

```
freq +s$gf% +s$gf:a
```

and your text has these codes

```
$gf $gf:a $gf:b $gf:c
```

your output will be

```
$gf%            3
$gf             1
```

Because $gf:a matches specifically to the +s$gf:a, it is excluded from matching +s$gf%.

One can also use the +s switch to remove certain strings from automatic exclusion. For example, the MLU program automatically excludes xxx, 0, uh, and words beginning with & from the MLU count. This can be changed by using this command:

```
mlu +s+uh +s+xxx +s+0* +s+&* file.cha
```

9.6 +T Option

This option allows you to include or exclude particular tiers. In CHAT formatted files, there exist three tier code types: main speaker tiers (denoted by *), speaker-dependent tiers (denoted by %), and header tiers (denoted by @). The speaker-dependent tiers are attached to speaker tiers. If, for example, you request to analyze the speaker *MOT and all the %cod dependent tiers, the programs will analyze all of the *MOT main tiers and only the %cod dependent tiers associated with that speaker.

The +t option allows you to specify which main speaker tiers, their dependent tiers, and header tiers should be included in the analysis. All other tiers, found in the given file, will be ignored by the program. For example, the command:

```
freq +t*CHI +t%spa +t%mor +t"@Group of Mot" sample.cha
```

tells FREQ to look at only the *CHI main speaker tiers, their %spa and %mor dependent tiers, and @Situation header tiers. When tiers are included, the analysis will be done on only those specified tiers.

The -t option allows you to specify which main speaker tiers, their dependent tiers, and header tiers should be excluded from the analysis. All other tiers found in the given file should be included in the analysis, unless specified otherwise by default. The command:

```
freq -t*CHI -t%spa -t%mor -t@"Group of Mot" sample.cha
```

tells FREQ to exclude all the *CHI main speaker tiers together with all their dependent tiers, the %spa and %mor dependent tiers on all other speakers, and all @Situation header tiers from the analysis. All remaining tiers will be included in the analysis.

When the transcriber has decided to use complex combinations of codes for speaker IDs such as *CHI-MOT for “child addressing mother,” it is possible to use the +t switch with the # symbol as a wildcard, as in these commands:

```
freq +t*CHI-MOT sample.cha
freq +t*#-MOT sample.cha
freq +t*CHI-# sample.cha
```

When tiers are included, the analysis will be done on only those specified tiers. When tiers are excluded, however, the analysis is done on tiers other than those specified. Failure to exclude all unnecessary tiers will cause the programs to produce distorted results. Therefore, it is safer to include tiers in analyses than to exclude them, because it is often difficult to be aware of all the tiers present in any given data file.

If only a tier-type symbol (*, %, @) is specified following the +t/-t options, the programs will include all tiers of that particular symbol type in the analysis. Using the option +t@ is important when using KWAL for limiting (see the description of the KWAL program), because it makes sure that the header information is not lost.

The programs search sequentially, starting from the left of the tier code descriptor, for exactly what the user has specified. This means that a match can occur wherever what has been specified has been found. If you specify *M on the command line after the option, the program will successfully match all speaker tiers that start with *M, such as *MAR, *MIK, *MOT, and so forth. For full clarity, it is best to specify the full tier name after the +t/-t options, including the : character. For example, to ensure that only the *MOT speaker tiers are included in the analysis, use the +t*MOT: notation.

As an alternative to specifying speaker names through letter codes, you can use the form:

```
+t@id=idcode
```

In this form, the "idcode" is any character string that matches the type of string that has been declared at the top of each file using the @ID header tier.

All of the programs include the main speaker tiers by default and exclude all of the dependent tiers, unless a +t% switch is used.

9.7 +U Option

This option merges specified files together. By default, when the user has specified a series of files on the command line, the analysis is performed on each individual file. The program then provides separate output for each data file. If the command line uses the +u option, the program combines the data found in all the specified files into one set and outputs the result for that set as a whole. If too many files are selected, CLAN may eventually be unable to complete this merger.

9.8 +V Option

This switch gives you the date when the current version of CLAN was compiled.

9.9 +W Option

This option controls the printing of additional sentences before and after a matched sentence. This option can be used with either KWAL or COMBO. These programs are used to display tiers that contain keywords or regular expressions as chosen by the user. By default, KWAL and COMBO combine the user-chosen main and dependent tiers into "clusters." Each cluster includes the main tier and its dependent tiers. (See the +u option for further information on clusters.)

The -w option followed by a positive integer causes the program to display that number of clusters before each cluster of interest. The +w option followed by a positive integer causes the program to display that number of clusters after each cluster of interest. For example, if you wanted the KWAL program to produce a context larger than a single cluster, you could include the -w3 and +w2 options in the command line. The program would then output three clusters above and two clusters below each cluster of interest.

9.10 +Y Option

This option allows you to work on non-CHAT files. Most of the programs are designed to work best on CHAT formatted data files. However, the +y option allows the user to use these programs on non-CHAT files. The program considers each line of a non-CHAT file to be one tier. There are two values of the +y switch. The +y value works on lines and the +y1 value works on utterances as delimited by periods, question marks, and exclamation marks. Some programs do not allow the use of the +y option at all. Workers interested in using CLAN with nonconversational data may wish to first convert there files to CHAT format using the TEXTIN program in order to avoid having to avoid the problematic use of the +y option.

9.11 +Z Option

This option allows the user to select any range of words, utterances, or speaker turns to be analyzed. The range specifications should immediately follow the option. For example:

+z10w	analyze the first ten words only.
+z10u	analyze the first ten utterances only.
+z10t	analyze the first ten speaker turns only.
+z10w-20w	analyze 11 words starting with the 10th word.
+z10u-20u	analyze 11 utterances starting with the 10th utterance.
+z10t-20t	analyze 11 speaker turns starting with the 10th turn.
+z10w-	analyze from the tenth word to the end of file.
+z10u-	analyze from the tenth utterance to the end of file.
+z10t-	analyze from the tenth speaker turn to the end of file.

If the +z option is used together with the +t option to select utterances from a particular speaker, then the counting will be based only on the utterances of that speaker. For example, this command:

```
mlu +z50u +t*CHI 0611.cha
```

will compute the MLU for the first 50 utterances produced by the child. If the +z option is used together with the +s option, the counting will be dependent on the working of the +s option and the results will seldom be as expected. To avoid this problem, you should use piping, as in this example:

```
kwal +d  +z1-3u +t*CHI sample.cha | kwal  +sMommy
```

If the user has specified more items than exist in the file, the program will analyze only the existing items. If the turn or utterance happens to be empty, because it consists of special symbols or words that have been selected to be excluded, then this utterance or turn is not counted.

The usual reason for selecting a fixed number of utterances is to derive samples that are comparable across sessions or across children. Often researchers have found that samples of 50 utterances provide almost as much information as samples of 100 utterances. Reducing the number of utterances being transcribed is important for clinicians who have been assigned a heavy case load.

You can use the +z switch with KWAL and pipe the results to a second program, rather than using it directly with FREQ or MLU. For example, in order to specifically exclude unintelligible utterances in an MLU analysis of the first 150 utterances from the Target Child, you could use this form:

```
kwal +z150u  +d +t*CHI 0042.cha -syyy -sxxx | mlu
```

You can also use postcodes to further control the process of inclusion or exclusion.

9.12 Metacharacters for Searching

Metacharacters are special characters used to describe other characters or groups of characters. Certain metacharacters may be used to modify search strings used by the +s/-s switch. However, in order to use metacharacters in the CHSTRING program a special switch must be set. The CLAN metacharacters are:

*	Any number of characters matched
%	Any number of characters matched and removed
%%	As above plus remove previous character
_	Any single character matched
\	Quote character

Suppose you would like to be able to find all occurrences of the word “cat” in a file. This includes the plural form “cats,” the possessives “cat-'s,” “cat-s'” and the contractions “cat-'is” and “cat-'has.” Using a metacharacter (in this case, the asterisk) would help you to find all of these without having to go through and individually specify each one. By inserting the string cat* into the include file or specifying it with +s option, all these forms would be found. Metacharacters can be placed anywhere in the word.

The * character is a wildcard character; it will find any character or group of continuous characters that correspond to its placement in the word. For example, if b*s were specified, the program would match words like “beads,” “bats,” “bat-'s,” “balls,” “beds,” “bed-s,” “breaks,” and so forth.

The % character allows the program to match characters in the same way that the * symbol does. Unlike the * symbol, however, all the characters matched by the % will be ignored in terms of the way of which the output is generated. In other words, the output will treat “beat” and “bat” as two occurrences of the same string, if the search string is b%t. Unless the % symbol is used with programs that produce a list of words matched by given keywords, the effect of the % symbol will be the same as the effect of the * symbol.

When the percentage symbol is immediately followed by a second percentage symbol, the effect of the metacharacter changes slightly. The result of such a search would be that the % symbol will be removed along with any one character preceding the matched string. Without adding the additional % character, a punctuation symbol preceding the wildcard string will not be matched ane will be ignored. Adding the second % sign can be particularly useful when searching for roots of words only. For example, to produce a word frequency count of the stem “cat,” specify this command:

```
freq +s"cat-%%" file.cha.
```

The first % sign matches the suffixes and the second one matches the dash mark. Thus, the search string specified by the +s option will match words like: “cat,” “cat-s,” “cat-'s,” and “cat-s” and FREQ will count all of these words as one word “cat.” If the data file file.cha had consisted of only those four words, the output of the FREQ program would have been: 4 cat. The limitation of this search is that it will not match words like “cats” or “cat's,” because the second percentage symbol is used to match the punctuation mark. The second percentage symbol is also useful for matching hierarchical codes such as $NIA:RP:IN.

The underline character _ is similar to the * character except that it is used to specify any single character in a word. For example, the string b_d will match words like “bad,” “bed,” “bud,” “bid,” and so forth. For detailed examples of the use of the percentage, underline, and asterisk symbols, see the section special characters.

The quote character (\) is used to indicate the quotation of one of the characters being used as metacharacters. Suppose that you wanted to search for the actual symbol (*) in a text. Because the (*) symbol is used to represent any character, it must be quoted by inserting the (\) symbol before the (*) symbol in the search string to represent the actual (*) character, as in “string*string.” To search for the actual character (\), it must be quoted also. For example, “string\\string” will match “string” followed by “\” and then followed by a second “string.”

10 MOR – Morphosyntactic Analysis

The modern study of child language development owes much to the methodological and conceptual advances introduced by Brown (1973). In his study of the language development of Adam, Eve, and Sarah, Roger Brown focused on a variety of core measurement issues, such as acquisition sequence, growth curves, morpheme inventories, productivity analysis, grammar formulation, and sampling methodology. The basic question that Brown was trying to answer was how one could use transcripts of interactions between children and adults to test theoretical claims regarding the child's learning of grammar. Like many other child language researchers, Brown considered the utterances produced by children to be a remarkably rich data source for testing theoretical claims. At the same time, Brown realized that one needed to specify a highly systematic methodology for collecting and analyzing these spontaneous productions.

Language acquisition theory has advanced in many ways since Brown (1973), but we are still dealing with many of the same basic methodological issues he confronted. Elaborating on Brown's approach, researchers have formulated increasingly reliable methods for measuring the growth of grammar, or morphosyntax, in the child. These new approaches serve to extend Brown's vision into the modern world of computers and computational linguistics. New methods for tagging parts of speech and grammatical relations now open up new and more powerful ways of testing hypotheses and models regarding children's language learning.

Before embarking on our review of computational tools in CHILDES, it is helpful to review briefly the ways in which researchers have come to use transcripts to study morphosyntactic development. When Brown collected his corpora back in the 1960s, the application of generative grammar to language development was in its infancy. However, throughout the 1980s and 1990s (Chomsky & Lasnik, 1993), linguistic theory developed increasingly specific proposals about how the facts of child language learning could illuminate the shape of Universal Grammar. At the same time, learning theorists were developing increasingly powerful methods for extracting linguistic patterns from input data. Some of these new methods relied on distributed neural networks (Rumelhart & McClelland, 1986), but others focused more on the ways in which children can pick up a wide variety of patterns in terms of relative cue validity (MacWhinney, 1987).

These two very different research traditions have each assigned a pivotal role to the acquisition of morphosyntax in illuminating core issues in learning and development. Generativist theories have emphasized issues such as: the role of triggers in the early setting of a parameter for subject omission (Hyams & Wexler, 1993), evidence for advanced early syntactic competence (Wexler, 1998), evidence for early absence functional categories that attach to the IP node (Radford, 1990), the role of optional infinitives in normal and disordered acquisition (Rice, 1997), and the child's ability to process syntax without any exposure to relevant data (Crain, 1991). Generativists have sometimes been criticized for paying inadequate attention to the empirical patterns of distribution in children's productions. However, work by researchers, such as

Stromswold (1994), van Kampen (1998), and Meisel (1986), demonstrates the important role that transcript data can play in evaluating alternative generative accounts.

Learning theorists have placed an even greater emphasis on the use of transcripts for understanding morphosyntactic development. Neural network models have shown how cue validities can determine the sequence of acquisition for both morphological (MacWhinney & Leinbach, 1991; MacWhinney, Leinbach, Taraban, & McDonald, 1989; Plunkett & Marchman, 1991) and syntactic (Elman, 1993; Mintz, Newport, & Bever, 2002; Siskind, 1999) development. This work derives further support from a broad movement within linguistics toward a focus on data-driven models (Bybee & Hopper, 2001) for understanding language learning and structure. These accounts formulate accounts that view constructions (Tomasello, 2003) and item-based patterns (MacWhinney, 1975) as the loci for statistical learning.

10.1 Analysis by Transcript Scanning

Although the CHILDES Project has succeeded in many ways, it has not yet provided a complete set of computational linguistic tools for the study of morphosyntactic development. In order to conduct serious corpus-based research on the development of morphosyntax, users will want to supplement corpora with tags that identify the morphological and syntactic status of every morpheme in the corpus. Without these tags, researchers who want to track the development of specific word forms or syntactic structures are forced to work with a methodology that is not much more advanced than that used by Brown in the 1960s. In those days, researchers looking for the occurrence of a particular morphosyntactic structure, such as auxiliary fronting in yes-no questions, would have to simply scan through entire transcripts and mark occurrences in the margins of the paper copy using a red pencil. With the advent of the personal computer in the 1980s, the marks in the margins were replaced by codes entered on a %syn (syntactic structure) or %mor (morphological analysis with parts of speech) coding tier. However, it was still necessary to pour over the full transcripts line by line to locate occurrences of the relevant target forms.

10.2 Analysis by Lexical Tracking

If a researcher is clever, there are ways to convince the computer to help out in this exhausting process of transcript scanning. An easy first step is to download the CLAN programs from the CHILDES website at http://childes.psy.cmu.edu. These programs provide several methods for tracing patterns within and between words. For example, if you are interested in studying the learning of English verb morphology, you can create a file containing all the irregular past tense verbs of English, as listed in the CHILDES manual. After typing all of these words into a file and then naming that file something like irreg.cut, you can use the CLAN program called KWAL with the +s@irreg.cut switch to locate all the occurrences of irregular past tense forms. Or, if you only want a frequency count, you can run FREQ with the same switch to get a frequency count of the various forms and the overall frequency of irregulars. Although this type of search is very helpful, you will also want to be able to search for overregularizations and overmarkings such as "*ranned", "*runned", "*goed", or "*jumpeded". Unless these are already specially marked in the transcripts, the only way to locate these forms is to create an even bigger list with all possible overmarkings. This is possible for the common irregular overmarkings, but doing this for all overmarked regulars, such as "*playeded", is not really possible. Finally, you also want to locate all correctly marked regular verbs. Here, again, making the search list is a difficult matter. You can search for all words ending in –ed, but you will have to cull out from this list forms like "bed", "moped", and "sled". A good illustration of research based on generalized lexical searches of this type can be found in the study of English verb learning by Marcus et al. (1992).

Or, to take another example, suppose you would like to trace the learning of auxiliary fronting in yes-no questions. For this, you would need to create a list of possible English auxiliaries to be included in a file called aux.cut. Using this, you could easily find all sentences with auxiliaries and then write out these sentences to a file for further analysis. However, only a minority of these sentences will involve yes-no questions. Thus, to further sharpen your analysis, you would want to further limit the search to sentences in which the auxiliary begins the utterance. To do this, you would need to dig carefully through the electronic version of the CHILDES manual to find the ways in which to use the COMBO program to compose search strings that include markers for the beginnings and ends of sentences. Also, you may wish to separate out sentences in which the auxiliary is moved to follow a wh-word. Here, again, you can compose a complicated COMBO search string that looks for a list of possible initial interrogative or "wh" words, followed by a list of possible auxiliaries. Although such searches are possible, they tend to be difficult, slow, and prone to error. Clearly, it would be better if the searches could examine not strings of words, but rather strings of morphosyntactic categories. For example, we would be able to trace sentences with initial wh-words followed by auxiliaries by just looking for the pattern of "int + aux". However, in order to perform such searches, we must first tag our corpora for the relevant morphosyntactic features. The current article explains how this is done.

10.3 Analysis by MOR, POST, and GRASP

So far, our analysis has examined how researchers can use the database through transcript scanning and lexical scanning. However, often these methods are inadequate for addressing broader and more complex issues such as detailed syntactic analysis or the comparisons and evaluations of full generative frameworks. To address these more complex issues, the CHILDES system now provides full support for analysis based on automatic morphosyntactic coding. The core programs used in this work are MOR, POST, and GRASP.

The initial morphological tagging of the CHILDES database relies on the application of the MOR program. Running MOR on a CHAT file is easy. In the simplest case, it involves nothing much more than a one-line command. However, before discussing the mechanics of MOR, let us take a look at what it produces. To give an example of the results of a MOR analysis for English, consider this sentence from eve15.cha in Roger Brown's corpus for Eve.

```
*CHI:  oop I spilled it .
```

%mor: int|oop pro|I v|spill-PAST pro|it .

Here, the main line gives the child’s production and the %mor line gives the part of speech for each word, along with the morphological analysis of affixes, such as the past tense mark (-PAST) on the verb. The %mor lines in these files were not created by hand. To produce them, we ran the MOR program, using the MOR grammar for English, which can be downloaded from http://childes.psy.cmu.edu/morgrams/.

The command for running MOR is nothing more in this case than “mor *.cha”. After running MOR, the file looks like this:

```
*CHI:   oop I spilled it .
%mor:   int|oop pro|I part|spill-PERF^v|spill-PAST pro|it .
```

Notice that the word “spilled” is initially ambiguous between the past tense and participle readings. To resolve such ambiguities, we run a program called POST. Running POST for English is also simple. The command is just “post *.cha” After POST has been run, the sentence is then “disambiguated.” Using this disambiguated form, we can then run the GRASP program, which is currently a separate program available from the CHILDES website, to create the representation given in the %xsyn line below:

```
*CHI:    oop I spilled it .
%mor:    int|oop pro|I v|spill-PAST pro|it .
%xsyn:  1|3|JCT 2|3|SUBJ 3|0|ROOT 4|3|OBJ 5|3|PUNCT
```

In this %xsyn line, we see that the second word “I” is related to the verb (“spilled”) through the grammatical relation (GR) of Subject. The fourth word “it” is related to the verb through the grammatical relation of Object.

Using GRASP, we have recently inserted dependency grammar tags for all of these grammatical relations in the Eve corpus. In tests run on the Eve corpus, 94% of the tags were assigned accurately (Sagae, Davis, Lavie, MacWhinney, & Wintner, 2007). A further test of GRASP on the Adam corpus also yielded an accuracy level of 94%. For both of these corpora, grammatical relations were mistagged 6% of the time. It is likely that, over time, this level of accuracy will improve, although we would never expect 100% accuracy for any tagging program. In fact, only a few human taggers can achieve 94% accuracy in their first pass tagging of a corpus.

The work of building MOR, POST, and GRASP has been supported by a number of people. Mitzi Morris built MOR in 1997, using design specifications from Roland Hausser. Since 2000, Leonid Spektor has extended MOR in many ways. Christophe Parisse built POST and POSTTRAIN (Parisse & Le Normand, 2000) and continues to maintain and refine them. Kenji Sagae built GRASP as a part of his dissertation work for the Language Technologies Institute at Carnegie Mellon University (Sagae, MacWhinney, & Lavie, 2004a, 2004b). GRASP was then applied in detail to the Eve and Adam corpus by Eric Davis and Shuly Wintner.

These initial experiments with GRASP and computational modelling of grammatical development in the CHILDES corpora underscore the increasing importance of methods from computational linguistics for the analysis of child language data. Together with statistical computational analyses (Edelman, Solan, Horn, & Ruppin, 2004) and neural network analyses (Li, Zhao, & MacWhinney, 2007), we should expect to see increasing input from computational linguistics, as the morphosyntactic tagging of the CHILDES database becomes increasingly refined and accurate.

The computational design of MOR was guided by Roland Hausser's (1990) MORPH systm and was implemented by Mitzi Morris. The system has been designed to maximize portability across languages, extendability of the lexicon and grammar, and compatibility with the CLAN programs. The basic engine of the parser is language independent. Language-specific information is stored in separate data files. The rules of the language are in data files that can be modified by the user. The lexical entries are also kept in ASCII files and there are several techniques for improving the match of the lexicon to a particular corpus. In order to avoid having too large a lexical file, only stems are stored in the lexicon and inflected forms appropriate for each stem are compiled at run time.

MOR automatically generates a %mor tier in which words are labeled by their syntactic category or "scat", followed by the pipe separator |, followed by the word itself, broken down into its constituent morphemes.

```
*CHI:	the people are making cakes .
%mor:	det|the n|people v:aux|be&PRES v|make-ING
		n|cake-PL .
```

The MOR program looks at each word on the main tier, without regard to context, and provides all possible grammatical categories and morphological analyses, as in the following example with the words "to" and "back." The caret ^ denotes the multiple possibilities for each word on the main tier.

```
*CHI:	I want to go back.
%mor:	pro|I v|want inf|to^prep|to
		v|go adv|back^n|back^v|back .
```

In order to select the correct form for each ambiguous case, the user can either edit the file using Disambiguator Mode or use POST.

One way of restricting the possible categories inserted by MOR is to use the replacement symbol [: text] on the main line for difficult cases. For example, the English form "wanna" could mean either "want to" or "want a". Similarly, "gotta" could be either "got to" or "got a." The transcriber can commit to one of these two readings on the main line by using this method:

```
*CHI:	I wanna [: want to] go back.
%mor:	pro|I v|want inf|to^prep|to v|go adv|back^n|back^v|back .
```

In this example, MOR will only attend to the material in the square brackets and will ignore the form "wanna."

10.4 Configuring MOR

For MOR to run successfully, you need to configure your grammar files and lexicon files into their proper positions in the MOR library directory. You will want to create a specific library directory for MOR that is distinct from the general CLAN **lib** directory. It is often convenient to place this MOR library inside the CLAN **lib** directory. In the MOR library directory, you need these three grammar files on top: ar.cut, cr.cut, and sf.cut. Optionally, you may also want to have a file called dr.cut. Within this directory, you then need to have a subdirectory called lex, which contains all of the various closed and open class lexicon files such as adj.cut, clo.cut, prep.cut, or n.cut. If you have retrieved the MOR grammar from the Internet or the CD-ROM, the materials will already be configured in the correct relative positions. Each separate grammar should be stored in its own folder and you should select the grammar you wish to use by setting the MORLIB location in the commands window.

10.4.1 Grammar and Lexicon Files

MOR relies on three files to specify the morphological processes of the language. They are:

1. **The allomorph rules file.** This file lists the ways in which morphemes vary in shape. The rules that describe these variations are called “arules.” The name of this file should be ar.cut.
2. **The concatenation rules file.** This file lists the ways in which morphemes can combine or concatenate. The rules that describe allowable concatenations are called “crules”. The name of this file should be cr.cut.
3. **The special form markers file.** The CHAT manual presents a series of special form markers that help identify lexical types such as neologisms, familial words, onomatopoeia, or second-language forms. MOR can use these markings to directly insert the corresponding codes for these words onto the %mor line. The sf.cut file includes all of these special form markers. In addition, these types must be listed in the first declaration in the cr.cut file. For English, all this is already done. If you are creating a grammar for another language, you can model your materials on the English example. The syntax of the lines in the sf.cut file is fairly simple. Each line has a special form marker, followed by the category information you wish to see inserted in the %mor line. If you wish to pull out capitalization words as being proper nouns, despite the shape of the special form marker, you can place \c to indicate uppercase before the special form marker. You must then add \l on another line to indicate what you want to have done with lowercase examples. See the English sf.cut file for examples.

In addition to these three grammar files, MOR uses a set of lexicon files to specify the shapes of individual words and affixes. These forms are stored in a group of files in the lexicon folder. The affix.cut file includes the prefixes and suffixes for the language. The other files contain the various open and closed class words of the language. At run time, MOR used the grammar rules to “blow up” the content of the lexicon files into a large binary tree that represents all the possible words of the language.

The first action of the parser program is to load the ar.cut file. Next the program reads in the files in your lexicon folder and uses the rules in ar.cut to build the run-time lexicon. If your lexicon files are fairly big, you will need to make sure that your machine has enough memory. On Macintosh, you can explicitly assign memory to the program. On Windows, you will have to make sure that your machine has lots of memory. Once the run-time lexicon is loaded, the parser then reads in the cr.cut file. Additionally, if the +b option is specified, the dr.cut file is also read in. Once the concatenation rules have been loaded the program is ready to analyze input words. As a user, you do not need to concern yourself about the run-time lexicon. Your main concern is about the entries in the lexicon files. The rules in the ar.cut and cr.cut files are only of concern if you wish to have a set of analyses and labelings that differs from the one given in the chapter of the CHAT manual on morphosyntactic coding, or if you are trying to write a new set of grammars for some language.

10.4.2 Disambiguation Rules

Although the MOR system relies primarily on POST for automatic disambiguation, it is sometimes useful to do some forced rule-based disambiguation during the development of a training corpus in preparation for the training of POSTTRAIN. To do this, you can use the +b switch in MOR to read a set of disambiguation rules that are kept in a file called dr.cut. Here are some sample disambiguation rules:

```
RULENAME: adj-n   % this rule chooses adj before nouns
choose
CURCAT = [scat adj]
when
NEXTCAT = [scat n]

RULENAME: asp   %  this rule chooses Chinese asp between verbs and nouns
choose
CURCAT = [scat asp]
when
PREVCAT = [scat v]
NEXTCAT = [scat n]

RULENAME: rel      % this rule chooses rel when it is after a verb and not sentence final
choose
CURCAT = [scat rel]
when
PREVCAT = [scat v]
NEXTCAT = ![end .]

RULENAME: rel      % this rule chooses pro:wh when it is sentence final
choose
CURCAT = [scat rel]
when
NEXTCAT = [end OR . ?]    % this could also be [end *] or even NEXTSURF = ?|.
```

10.4.3 Unique Options

+b Use the dr.cut disambiguation rules.

+c With this option, clitics such as 'd, n't , and 'll will be treated as separate words. This option must be used when creating the %mor tier for DSS analysis.

+eS Show the result of the operation of the arules on either a stem S or stems in file @S. This output will go into a file called debug.cdc in your library directory. Another way of achieving this is to use the +d option inside "interactive MOR"

+xi Run MOR in the interactive test mode. You type in one word at a time to the test prompt and MOR provides the analysis on line. This facility makes the following commands available in the CLAN Output window:

```
word - analyze this word
:q  quit- exit program
:c  print out current set of crules
:d  display application of arules.
:l  re-load rules and lexicon files
:h  help - print this message
```

If you type in a word, such as "dog" or "perro," MOR will try to analyze it and give you its components morphemes. If you change the rules or the lexicon, use :l to reload and retest. The :c and :d switches will send output to a file called debug.cdc in your library directory.

+xl Run MOR in the lexicon building mode. This mode takes a series of .cha files as input and outputs a small lexical file with the extension .ulx with entries for all words not recognized by MOR. This helps in the building of lexicons.

MOR also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

10.4.4 MOR **Lexicons**

Before running MOR on a set of CHAT files, it is important to make sure that MOR will be able to recognize all the words in these files. A first step in this process involves running the CHECK program to make sure that all of the words follow basic CHAT rules, such as not including numbers or capital letters in the middle of words. For more details about these various CHECK requirements, please consult the sections of this manual that describe CHECK.

Once you know that a corpus passes CHECK, you will want to see whether it contains words that are not yet in the MOR lexicon. You can do this quickly by running the command

```
mor +xl *.cha
```

and checking the output file which will list all the words not yet recognized by MOR. It is extremely unlikely that every word in any large corpus of child language data would be listed in even the largest MOR lexicon. Therefore, users of MOR need to understand how to supplement the basic lexicons with additional entries. Before we look at the process of adding new words to the lexicon, we first need to examine the way in which entries in the disk lexicon are structured.

The disk lexicon contains truly irregular forms of a word as well as citation forms. For example, the verb “go” is stored in the disk lexicon, along with the past tense “went,” since this latter form is suppletive and does not undergo regular rules. The disk lexicon contains any number of lexical entries, stored at most one entry per line. The lexicon may be annotated with comments, which will not be processed. A comment begins with the percent sign and ends with a new line. A lexical entry consists of these parts:

1. The surface form of the word.
2. Category information about the word, expressed as a set of feature-value pairs. Each feature-value pair is enclosed in square brackets and the full set of feature-value pairs is enclosed in curly braces. All entries must contain a feature-value pair that identifies the syntactic category to which the word belongs, consisting of the feature “scat” with an appropriate value.
3. Following the category information is information about the lemmatization of irregular forms. This information is given by having the citation form of the stem followed by the & symbol as the morpheme separator and then the grammatical morphemes it contains.
4. Finally, if the grammar is for a language other than English, you can enter the English translation of the word preceded by the = sign.

The following are examples of lexical entries:

```
can     {[scat v:aux]}
a       {[scat det]}
an      {[scat det]}      "a"
go      {[scat v] [ir +]}
went    {[scat v] [tense past]}       "go&PAST"
```

When adding new entries to the lexicon it is usually sufficient to enter the citation form of the word, along with the syntactic category information, as in the illustration for the word “a” in the preceding examples. When working with languages other than English, you may wish to add English glosses and even special character sets to the lexicon. For example, in Cantonese, you could have this entry:

```
ping4gwo2       {[scat n]} =apple
```

To illustrate this, here is an example of the MOR output for an utterance from Cantonese:

```
*CHI:   sik6 ping4gwo2 caang2 hoeng1ziu1 .
%mor:   v|sik6=eat n|ping4gwo2=apple
        n|caang2=orange n|hoeng1ziu1=banana .
```

In languages that use both Roman and non-Roman scripts, such as Chinese, you may also want to add non-Roman characters after the English gloss. This can be done using this form in which the $ sign separates the English gloss from the representation in characters:

```
pinyin  {[scat x]} “lemmatization” =gloss$characters=
```

MOR will take the forms indicated by the lemmatization, the gloss, and the characters and append them after the category representation in the output. The gloss should not contain spaces or the morpheme delimiters +, -, and #. Instead of spaces or the + sign, you can use the underscore character to represent compounds.

10.4.5 Lexicon Building

Once the file is thoroughly CHECK-ed, you are ready to make a first run of MOR to see how many words in your files are not yet in the MOR lexicon. The command is simply:

```
mor +xl *.mor
```

When MOR is run with the +xl flag, the output is a single “minilex” file with the extension .ulx which contains templates for the lexical entries for all unknown words in a collection of files. Duplicates are removed automatically when MOR creates the .ulx file. A fragment of the output of this command might look something like this:

```
ta              {[scat ?]}
tag             {[scat ?]}
tags            {[scat ?]}
talkative       {[scat ?]}
tambourine      {[scat ?]}
```

You must then go through this file and determine whether to discard, complete, or modify these entry templates. For example, it may be impossible to decide what category “ta” belongs to without examining where it occurs in the corpus. In this example, a scan of the Sarah files in the Brown corpus (from which these examples were taken), reveals that “ta” is a variant of the infinitive marker “to”:

```
*MEL:  yeah # (be)cause if it's gon (t)a be a p@l it's
        got ta go that way.
```

Therefore, the entry for “ta” is amended to:

```
ta     {[scat inf]}     "to"
```

The corpus includes both the form “tag” and “tags.” However, because the former can be derived from the latter, it is sufficient to have just the entry for “tag” in the lexicon. The forms “talkative” and “tambourine” are low-frequency items that are not included in the standard lexicon file eng.lex. Inasmuch as these are real words, the ? should be replaced by the codes “adj” and “n”, respectively. For the example fragment given above, the resulting .ulx file should look like this:

```
ta              {[scat inf]} "to"
tag             {[scat n]}
talkative       {[scat adj]}
tambourine      {[scat n]}
```

Once all words have been coded, you need to insert each new word into one of the lexicon files. If you do not want to edit the main files, you can create new ones such as adj2.cut for all your new adjectives or vir2.cut for additional irregular verbs.

10.4.6 A Formal Description of the Rule Files

Users working with languages for which grammar files have already been built do not need to concern themselves with the remaining sections on MOR. However, users who need to develop grammars for new languages or who find they have to modify grammars for existing ones will need to understand how to create the two basic rule files themselves. You do not need to create a new version of the sf.cut file for special form markers. You just copy this file and give it a name such as dansf.cut, if the prefix you want to use for your language is something like "dan" for Danish.

In order to build new versions of the arules and crules files for your language, you will need to study the English files or files for a related language. For example, when you are building a grammar for Portuguese, it would be helpful to study the grammar that has already been constructed for Spanish. This section will help you understand the basic principles underlying the construction of the arules and crules.

Declarative structure

Both arules and crules are written using a simple declarative notation. The following formatting conventions are used throughout:

1. Statements are one per line. Statements can be broken across lines by placing the continuation character \ at the end of the line.
2. Comments begin with a % character and are terminated by the new line. Comments may be placed after a statement on the same line, or they may be placed on a separate line.
3. Names are composed of alphanumeric symbols, plus these characters:
   ^ & + - _ : \ @ . /

Both arule and crule files contain a series of rules. Rules contain one or more clauses, each of which is composed of a series of **condition** statements, followed by a series of **action** statements. In order for a clause in rule to apply, the input(s) must satisfy all condition statements. The output is derived from the input via the sequential application of all the action statements.

Both condition and action statements take the form of equations. The left hand side of the equation is a keyword, which identifies the part of the input or output being processed. The right hand side of the rule describes either the surface patterns to be matched or generated, or the category information that must be checked or manipulated.

The analyzer manipulates two different kinds of information: information about the surface shape of a word, and information about its category. All statements that match or manipulate category information must make explicit reference to a feature or features. Similarly, it is possible for a rule to contain a literal specification of the shape of a stem or affix. In addition, it is possible to use a pattern matching language in order to give a more general description of the shape of a string.

Pattern-matching symbols

The specification of orthographic patterns relies on a set of symbols derived from the regular expression (regexp) system in Unix. The rules of this system are:

1. The metacharacters are: * [ ] | . ! All other characters are interpreted literally.
2. A pattern that contains no metacharacters will only match itself, for example the pattern “abc” will match only the string “abc”.
3. The period matches any character.
4. The asterisk * allows any number of matches (including 0) on the preceding character. For example, the pattern '.*' will match a string consisting of any number of characters.
5. The brackets [ ] are used to indicate choice from among a set of characters. The pattern [ab] will match either a or b.
6. A pattern may consist of a disjunctive choice between two patterns, by use of the | symbol. For example, the pattern will match all strings which end in x, s, sh, or ch.
7. It is possible to check that some input does not match a pattern by prefacing the entire pattern with the negation operator !.

Variable notation

A variable is used to name a regular expression and to record patterns that match it. A variable must first be declared in a special variable declaration statement. Variable declaration statements have the format: “VARNAME = regular-expression” where VARNAME is at most eight characters long. If the variable name is more than one character, this name should be enclosed in parenthesis when the variable is invoked. Variables are particularly important for the arules in the ar.cut file. In these rules, the negation operator is the up arrow ^, not the exclamation mark. Variables may be declared through combinations of two types of disjunction markers, as in this example for the definition of a consonant cluster in the English ar.cut file:

```
O = [^aeiou]|[^aeiou][^aeiou]|[^aeiou][^aeiou][^aeiou]|qu|squ
```

Here, the square brackets contain the definition of a consonant as not a vowel and the bar or turnstile symbols separate alternative sequences of one, two, or three consonants. Then, for good measure, the patterns “qu” and “squ” are also listed as consonantal onsets. For languages that use combining diacritics and other complex symbols, it is best to use the turnstile notation, since the square bracket notation assumes single characters. In these strings, it is important not to include any spaces or tabs, since the presence of a space will signal the end of the variable.

Once declared, the variable can be invoked in a rule by using the operator $. If the variable name is longer than a single character, the variable name should be enclosed in parentheses when invoked. For example, the statement `X = .*` declares and initializes a variable named “X.” The name X is entered in a special variable table, along with the regular expression it stands for. Note that variables may not contain other variables.

The variable table also keeps track of the most recent string that matched a named pattern. For example, if the variable X is declared as above, then the pattern $Xle will match all strings that end in "le". In particular, the string "able" will match this pattern; "ab" will match the pattern named by "X", and "le" will match the literal string "le". Because the string "ab" is matched against the named pattern X, it will be stored in the variable table as the most recent instantiation of X, until another string matches X.

Category Information Operators

The following operators are used to manipulate category information: ADD [feature value], and DEL [feature value]. These are used in the category action statements. For example, the crule statement "RESULTCAT = ADD [num pl]" adds the feature value pair [num pl] to the result of the concatenation of two morphemes.

Arules

The function of the arules is to expand the entries in the disk lexicon into a larger number of entries in the on-line lexicon. Words that undergo regular phonological or orthographic changes when combined with an affix only need to have one disk lexicon entry. The arules are used to create on-line lexicon entries for all inflectional variants. These variants are called **allos**. For example, the final consonant of the verb "stop" is doubled before a vowel-initial suffix, such as "-ing." The disk lexicon contains an entry for "stop," whereas the online lexicon contains two entries: one for the form "stop" and one for the form "stopp".

An arule consists of a header statement, which contains the rulename, followed by one or more condition-action **clauses**. Each clause has a series of zero or more conditions on the input, and one or more sets of actions. Here is an example of a typical condition-action clause from the larger n-allo rule in the English ar.cut file:

```
LEX-ENTRY:
LEXSURF = $Yy
LEXCAT = [scat n]
ALLO:
ALLOSURF = $Yie
ALLOCAT = LEXCAT, ADD [allo nYb]
ALLO:
ALLOSURF = LEXSURF
ALLOCAT = LEXCAT, ADD [allo nYa]
```

This is a single condition-action clause, labeled by the header statement "LEX-ENTRY:" Conditions begin with one of these two keywords:

1. LEXSURF matches the surface form of the word in the lexical entry to an abstract pattern. In this case, the variable declaration is
   Y = .*[^aeiou]
   Given this, the statement "LEXSURF = $Yy" will match all lexical entry surfaces that have a final y preceded by a nonvowel.
   3. LEXCAT checks the category information given in the matched lexical item against a given series of feature value pairs, each enclosed in square brackets and separated by commas. In this case, the rule is meant to apply

> only to nouns, so the category information must be [scat n]. It is possible to check that a feature-value pair is not present by prefacing the feature-value pair with the negation operator !.

Variable declarations should be made at the beginning of the rule, before any of the condition-action clauses. Variables apply to all following condition-action clauses inside a rule, but should be redefined for each rule.

After the condition statements come one or more action statements with the label ALLO: In most cases, one of the action statements is used to create an allomorph and the other is used to enter the original lexical entry into the run-time lexicon. Action clauses begin with one of these three keywords:

1. ALLOSURF is used to produce an output surface. An output is a form that will be a part of the run-time lexicon used in the analysis. In the first action clause, a lexical entry surface form like "pony" is converted to "ponie" to serve as the stem of the plural. In the second action clause, the original form "pony" is kept because the form "ALLOSURF = LEXSURF" causes the surface form of the lexical entry to be copied over to the surface form of the allo.
2. ALLOCAT determines the category of the output allos. The statement "ALLOCAT = LEXCAT" causes all category information from the lexical entry to be copied over to the allo entry. In addition, these two actions add the morphological classes such as [allo nYa] or [allo nYb] in order to keep track of the nature of these allomorphs during the application of the crules.
    4. ALLOSTEM is used to produce an output stem. This action is not necessary in this example, because this rule is fully regular and produces a noninflected stem. However, the arule that converts "postman" into "postmen" uses this ALLOSTEM action:

    ALLOSTEM = $Xman&PL

    The result of this action is the form postman&PL that is placed into the %mor line without the involvement of any of the concatenation rules.

There are two special feature types that operate to dump the contents of the arules and the lexicon into the output. These are "gen" and "proc". The gen feature introduces its value as a component of the stem. Thus the entry [gen masc] for the Spanish word "hombre" will end up producing n|hombre&MASC. The entry [proc dim] for Chinese reduplicative verbs wil end up producing v|kan4-DIM for the reduplicated form kan4kan4. These methods allow allorules to directly influence the output of MOR.

Every set of action statements leads to the generation of an additional allomorph for the online lexicon. Thus, if an arule clause contains several sets of action statements, each labeled by the header ALLO:, then that arule, when applied to one entry from the disk lexicon, will result in several entries in the online lexicon. To create the online lexicon, the arules are applied to the entries in the disk lexicon. Each entry is matched against the arules in the order in which they occur in the arules file. This ordering of arules is an extremely important feature. It means that you need to order specific cases before general cases to avoid having the general case preempt the specific case.

As soon as the input matches all conditions in the condition section of a clause, the actions are applied to that input to generate one or more allos, which are loaded into the on-line lexicon. No further rules are applied to that input, and the next entry from the disk lexicon is then read in to be processed. The complete set of arules should always end with a default rule to copy over all remaining lexical entries that have not yet been matched by some rule. This default rule must have this shape:

```
% default rule- copy input to output
RULENAME: default
LEX-ENTRY:
ALLO:
```

Crules

The purpose of the crules is to allow stems to combine with affixes. In these rules, sets of conditions and actions are grouped together into **if then** clauses. This allows a rule to apply to a disjunctive set of inputs. As soon as all the conditions in a clause are met, the actions are carried out. If these are carried out successfully the rule is considered to have “fired,” and no further clauses in that rule will be tried.

There are two inputs to a crule: the part of the word identified thus far, called the “start,” and the next morpheme identified, called the “next.” The best way to think of this is in terms of a bouncing ball that moves through the word, moving items from the not-yet-processed chunk on the right over to the already processed chunk on the left. The output of a crule is called the “result.” The following is the list of the keywords used in the crules:

condition keywords	*function*
STARTSURF	check surface of start input against some pattern
STARTCAT	check start category information
NEXTSURF	check surface of next input against some pattern
NEXTCAT	check next category information
MATCHCAT	check that start and next have the same value for all the feature-value pairs of the type specified
RESULTCAT	output category information

Here is an example of a piece of a rule that uses most of these keywords:

```
 S = .*[sc]h|.*[zxs] % strings that end in affricates
 O = .*[^aeiou]o % things that end in o
 % clause 1 - special case for "es" suffix
  if
  STARTSURF = $S
  NEXTSURF = es|-es
  NEXTCAT = [scat vsfx]
  MATCHCAT [allo]
  then
  RESULTCAT = STARTCAT, NEXTCAT [tense], DEL [allo]
```

```
RULEPACKAGE = ()
```

This rule is used to analyze verbs that end in -es. There are four conditions that must be matched in this rule:

1. The STARTSURF is a stem that is specified in the declaration to end in an affricate. The STARTCAT is not defined.
2. The NEXTSURF is the -es suffix that is attached to that stem.
3. The NEXTCAT is the category of the suffix, which is “vsfx” or verbal suffix.
    5. The MATCHCAT [allo] statement checks that both the start and next inputs have the same value for the feature allo. If there are multiple [allo] entries, all must match.

The shape of the result surface is simply the concatenation of the start and next surfaces. Hence, it is not necessary to specify this via the crules. The category information of the result is specified via the RESULTCAT statement. The statement “RESULTCAT = STARTCAT” causes all category information from the start input to be copied over to the result. The statement “NEXTCAT [tense]” copies the tense value from the NEXT to the RESULT and the statement “DEL [allo]” deletes all the values for the category [allo].

In addition to the condition-action statements, crules include two other statements: the CTYPE statement, and the RULEPACKAGES statement. The CTYPE statement identifies the kind of concatenation expected and the way in which this concatenation is to be marked. This statement follows the RULENAME header. There are two special CTYPE makers: START and END. “CTYPE: START” is used for those rules that execute as soon as one morpheme has been found. “CTYPE: END” is used for those rules that execute when the end of the input has been reached. Otherwise, the CYTPE marker is used to indicate which concatenation symbol is used when concatenating the morphemes together into a parse for a word. According to CLAN conventions, # is used between a prefix and a stem, - is used between a stem and suffix, and ~ is used between a clitic and a stem. In most cases, rules that specify possible suffixes will start with CTYPE: -. These rules are neither start nor end rules and they insert a suffix after the stem.

Rules with CTYPE START are entered into the list of startrules. Startrules are the set of rules applied as soon as a morpheme has been recognized. In this case, the beginning of the word is considered as the start input, and the next input is the morpheme first recognized. As the start input has no surface and no category information associated with it, conditions and actions are stated only on the next input.

Rules with CTYPE END are entered into the list of endrules. These rules are invoked when the end of a word is reached, and they are used to rule out spurious parses. For the endrules, the start input is the entire word that has just been parsed, and there is no next input. Thus conditions and actions are only stated on the start input.

The RULEPACKAGES statement identifies which rules may be applied to the result of a rule, when that result is the input to another rule. The RULEPACKAGES statement follows the action statements in a clause. There is a RULEPACKAGES statement associated with each clause. The rules named in a RULEPACKAGES statement are not tried until after another morpheme has been found. For example, in parsing the input "walking", the parser first finds the morpheme "walk," and at that point applies the startrules. Of these startrules, the rule for verbs will be fired. This rule includes a RULEPACKAGES statement specifying that the rule which handles verb conjugation may later be fired. When the parser has further identified the morpheme "ing," the verb conjugation rule will apply, where "walk" is the start input, and "ing" is the next input.

Note that, unlike the arules which are strictly ordered from top to bottom of the file, the crules have an order of application that is determined by their CTYPE and the way in which the RULEPACKAGES statement channels words from one rule to the next.

10.4.7 Interactive Mode

When building a grammar for a new language, it is best to begin with a paper-and-pencil analysis of the morphological system in which you lay out the various affixes of the language, the classes of stem allomorphy variations, and the forces that condition the choices between allomorphs. This work should be guided by a good descriptive grammar of the morphology of the language. Once this work is finished, you should create a small lexicon of the most frequent words. You may want to focus on one part-of-speech at a time. For example, you could begin with the adverbs, since they are often monomorphemic. Then you could move on to the nouns. The verbs should probably come last. You can copy the sf.cut file from English and rename it.

Once you have a simple lexicon and a set of rule files, you will begin a long process of working with interactive MOR. When using MOR in the +xi or interactive mode, there are several additional options that become available in the CLAN Output window. They are:

```
word - analyze this word
:q  quit- exit program
:c  print out current set of crules
:d  display application of a rules.
:l  re-load rules and lexicon files
:h  help - print this message
```

If you type in a word, such as "dog" or "perro," MOR will try to analyze it and give you its component morphemes. If all is well, you can move on the next word. If it is not, you need to change your rules or the lexicon. You can stay within CLAN and just open these using the Editor. After you save your changes, use :l to reload and retest.

When you begin work with the grammar, you want to focus on the use of the +xi switch, rather than the analysis of large groups of files. As you begin to elaborate your grammar, you will want to start to work with sets of files. These can be real data files or else files full of test words. When you shift to working with files, you will be combining the use of interactive MOR and the +xi switch with use of the lexicon testing facility that uses +xl. As you move through this work, make copies of your MOR grammar files and lexicon frequently, because you will sometimes find that you have made a change that makes everything break and you will need to go back to an earlier stage to figure out what you need to fix. We also recommend using a fast machine with lots of memory.

You will find that you are frequently reloading the grammar using the :l function. Having a fast machine will greatly speed this process.

To begin the process, start working with the sample minimal MOR grammars available from the net. These files should allow you to build up a lexicon of uninflected stems. Try to build up separate files for each of the parts of speech in your language. As you start to feel comfortable with this, you should begin to add affixes. To do this, you need to create a lexicon file, such as aff.cut. Using the technique found in unification grammars, you want to set up categories and allos for these affixes that will allow them to match up with the right stems when the crules fire. For example, you might want to call the plural a [scat nsfx] in order to emphasize the fact that it should attach to nouns. And you could give the designation [allo mdim] to the masculine diminutive suffix -ito in Spanish in order to make sure that it only attaches to masculine stems and produces a masculine output.

As you progress with your work, continually check each new rule change by entering :l (colon followed by "l" for load) into the CLAN Output window. If you have changed something in a way that produces a syntactic violation, you will learn this immediately and be able to change it back. If you find that a method fails, you should first rethink your logic. Consider these factors:

1. Arules are strictly ordered. Maybe you have placed a general case before a specific case.
2. Crules depend on direction from the RULEPACKAGES statement.
3. There has to be a START and END rule for each part of speech. If you are getting too many entries for a word, maybe you have started it twice. Alternatively, you may have created too many allomorphs with the arules.
4. If you have a MATCHCAT allos statement, all allos must match. The operation DEL [allo] deletes all allos and you must add back any you want to keep.
5. Make sure that you understand the use of variable notation and pattern matching symbols for specifying the surface form in the arules.

However, sometimes it is not clear why a method is not working. In this case, you will want to check the application of the crules using the :c option in the CLAN Output window. You then need to trace through the firing of the rules. The most important information is often at the end of this output.

If the stem itself is not being recognized, you will need to also trace the operation of the arules. To do this, you should either use the +e option in standard MOR or else the :d option in interactive MOR. The latter is probably the most useful. To use this option, you should create a directory called testlex with a single file with the words your are working with. Then run:

```
mor +xi +ltestlex
```

Once this runs, type :d and then :l and the output of the arules for this test lexicon will go to debug.cdc. Use your editor to open that file and try to trace what is happening there.

As you progress with the construction of rules and the enlargement of the lexicon, you can tackle whole corpora. At this point you will occasionally run the +xl analysis. Then you take the problems noted by +xl and use them as the basis for repeated testing using the +xi switch and repeated reloading of the rules as you improve them. As you build up your rule sets, you will want to annotate them fully using comments preceded by the % symbol.

10.4.8 Disambiguator Mode

Disambiguation is a special facility that is used to "clean up" the ambiguities in the %mor tier that are created by MOR. If you use the POST program, hand disambiguation is not necessary. However, when developing POST for a new language, you may find this tool useful. Toggling the **Disambiguator Mode** option in the **Mode** menu allows you to go back and forth between Disambiguator Mode and standard Editor Mode. In Disambiguator Mode, you will see each ambiguous interpretation on a %mor line broken into its alternative possibilities at the bottom of the editor screen. The user double-clicks on the correct option and it is inserted. An ambiguous entry is defined as any entry that has the ^ symbol in it. For example, the form N|back^Prep|back is ambiguously either the noun "back" or the preposition "back."

By default, Disambiguator Mode is set to work on the %mor tier. However, you may find it useful for other tiers as well. To change its tier setting, select the **Edit** menu and pull down to **Options** to get the **Options** dialog box. Set the disambiguation tier to the tier you want to disambiguate. To test all of this out, edit the sample.cha file, reset your default tier, and then type *Esc-2*. The editor should take you to the second %spa line which has:

```
%spa:   $RES:sel:ve^$DES:tes:ve
```

At the bottom of the screen, you will have a choice of two options to select. Once the correct one is highlighted, you hit a carriage return and the correct alternative will be inserted. If you find it impossible to decide between alternative tags, you can select the UND or undecided tag, which will produce a form such as "und|drink" for the word drink, when you are not sure whether it is a noun or a verb.

10.5 The Workings of MOR

When MOR runs, it breaks up morphemes into their component parts. In a relatively analytic language like English, many words require no analysis at all. However, even in English, a word like "coworkers" can be seen to contain four component morphemes, including the prefix "co", the stem, the agential suffix, and the plural. For this form, MOR will produce the analysis: co#n:v|work-AGT-PL. This representation uses the symbols # and – to separate the four different morphemes. Here, the prefix stands at the beginning of the analysis, followed by the stem (n|work), and the two suffixes. In general, stems always have the form of a part of speech category, such as "n" for noun, followed by the vertical bar and then a statement of the stem's lexical form.

In order to understand the functioning of the MOR grammar for English, the best place to begin is with a tour of the files inside the /english folder that you can download from the server. At the top level, you will see these files:

1. ar.cut – These are the rules that generate allomorphic variants from stems.
2. cr.cut – These are the rules that specify the possible combinations of morphemes going from left to right in an English word.
3. debug.cdc – This file holds the complete trace of an analysis of a given word by MOR. It always holds the results of the most recent analysis. It is mostly useful for people who are developing new ar.cut or cr.cut files as a way of tracing out or debugging problems with these rules.
4. docs – This is a folder containing a file of instructions on how to train POST and a list of tags and categories used in the English grammar.
5. eng.db – This is a file used by POST and should be left untouched.
6. ex.cut – This file includes analyses that are being “overgenerated” by MOR and should simply be filtered out or excluded whenever they occur.
7. lex – This is the heart of the MOR grammar. We will examine it in greater detail below.
8. posttags.cut – This is a list of the grammatical tags of English that should be included whenever running POST.
9. sf.cut – This file tells MOR how to deal with words that end with certain special form markers such as @b for babbling.
10. traintags.cut – This is a list of the tags that are used by POSTTRAIN when it creates the eng.db database.

When examining these files and others, please note that the exact shapes of the files, the word listings, and the rules will change over time. We recommend that users open up these various files to understand their contents. However, over time, the contents will diverge more and more from the names and examples given here. Still, it should be possible to trace through the same basic principles, even given these inevitable changes.

Now let us take a look at the files contained inside the /lex folder. Here, we find 72 files that break out the possible words of English into different files for each specific part of speech or compound structure. Because these distinctions are so important to the correct transcription of child language and the correct running of MOR, it is worthwhile to consider the contents of each of these various files. As the following table shows, about half of these word types involve different part of speech configurations within compounds. This analysis of compounds into their part of speech components is intended to further study of the child’s learning of compounds as well as to provide good information regarding the part of speech of the whole. The name of the compound files indicates their composition. For example the name adj+n+adj.cut indicates compounds with a noun followed by an adjective (n+adj) whose overall function is that of an adjective. In English, the part of speech of a compound is usually the same as that of the last component of the compound.

File	Function	Example
0affix.cut	prefixes and suffixes	see expanded list below
0uk.cut	terms local to the UK	fave, doofer, sixpence
adj-dup.cut	baby talk doubles	nice+nice, pink+pink
adj-ir.cut	irregular adjectives	better, furthest
adj-kidy.cut	adjectives with babytalk –y	bunchy, eaty, crawly
adj.cut	regular adjectives	tall, redundant
adj+adj+adj.cut	compounds	half+hearted, hot+crossed
adj+adj+adj(on).cut	compounds	super+duper, easy+peasy
adj+n+adj.cut	compounds	dog+eared, stir+crazy
adj+v+prep+n.cut	compounds	pay+per+view
adj+v+v.cut	compounds	make+believe, see+through
adv-int.cut	intensifying adverbs	really, plumb, quite
adv-loc.cut	locative adverbs	north, upstairs
adv-tem.cut	temporal adverbs	tomorrow, tonight, anytime
adv.cut	regular adverbs	ajar, fast, mostly
adv+adj+adv.cut	compounds	half+off, slant+wise
adv+adj+n.cut	compounds	half+way, off+shore
adv+n+prep+n.cut	compounds	face+to+face
auxil.cut	auxiliaries and modals	should, can, are
co-cant.cut	Cantonese bilngual forms	wo, wai, la
co-voc.cut	vocative communicators	honey, dear, sir
co.cut	regular communicators	blah, bybye, gah, no
conj.cut	conjunctions	and, although, because
det.cut	deictic determiners	this, that, the,
fil.cut	fillers	um, uh, er
int-rhymes.cut	rhymes as interjections	fee_figh_foe_fum
int.cut	interjections	farewell, boohoo, hubba
int+int+int.cut	compounds	good+afternoon
int+int+int+int.cut	compounds	ready+set+go
n-abbrev.cut	abbreviations	c_d, t_v, w_c
n-baby.cut	babytalk forms	passie, wawa, booboo
n-dashed.cut	non-compound combinations	cul_de_sac, seven_up
n-dup.cut	duplicate nouns	cow+cow, chick_chick
n-ir.cut	irregular nouns	children, cacti, teeth
n-loan.cut	loan words	goyim, amigo, smuck
n-pluraletant.cut	nouns with no singular	golashes, kinesics, scissors
n.cut	regular nouns	dog, corner, window
n+adj+n.cut	compounds	big+shot, cutie+pie
n+adj+v+adj.cut	compounds	merry+go+round
n+n+conj+n.cut	compounds	four+by+four, dot+to+dot
n+n+n-on.cut	compounds	quack+duck, moo+cow
n+n+n.cut	compounds	candy+bar, foot+race
n+n+novel.cut	compounds	children+bed, dog+fish
n+n+prep+det+n.cut	compounds	corn+on+the+cob

n+on+on-baby.cut	compounds	wee+wee, meow+meow
n+v+x+n.cut	compounds	jump+over+hand
n+v+n.cut	compounds	squirm+worm, snap+bead
n+v+ptl.cut	compounds	chin+up, hide+out
num-ord.cut	ordinals	fourth, thirteenth
num.cut	cardinals	five, twenty
on.cut	onomatopoeia	boom, choo_choo
on+on+on.cut	compounds	cluck+cluck, knock+knock
prep.cut	prepositions	under, minus
pro.cut	pronouns	he, nobody, himself
ptl.cut	verbal particle	up, about, on
quan.cut	quantifier	some, all, only, most
small.cut	assorted forms	not, to, xxx, yyy
v-baby.cut	baby verbs	wee, poo
v-clit.cut	cliticized forms	gonna, looka
v-dup.cut	verb duplications	eat+eat, drip+drip
v-ir.cut	irregular verbs	came, beset, slept
v.cut	regular verbs	run, take, remember
v+adj+v.cut	compounds	deep+fry, tippy+toe
v+n+v.cut	compounds	bunny+hop, sleep+walk
v+v+conj+v.cut	compounds	hide+and+seek
wh.cut	interrogatives	which, how, why
zero.cut	omitted words	0know, 0conj, 0is

The construction of these lexicon files involves a variety of decisions. Here are some of the most important issues to consider.

1. Words may often appear in several files. For example, virtually every noun in English can also function as a verb. However, when this function is indicated by a suffix, as in "milking" the noun can be recognized as a verb through a process of morphological derivation contained in a rule in the cr.cut file. In such cases, it is not necessary to list the word as a verb. Of course, this process fails for unmarked verbs. However, it is generally not a good idea to represent all nouns as verbs, since this tends to overgenerate ambiguity. Instead, it is possible to use the POSTMORTEM program to detect cases where nouns are functioning as bare verbs.
2. If a word can be analyzed morphologically, it should not be given a full listing. For example, since "coworker" can be analyzed by MOR into three morphemes as co#n:v|work-AGT, it should not be separately listed in the n.cut file. If it is, then POST will not be able to distinguish co#n:v|work-AGT from n|coworker.
3. In the zero.cut file, possible omitted words are listed without the preceding 0. For example, there is an entry for "conj" and "the". However, in the transcript, these would be represented as "0conj" and "0the".
4. It is always best to use spaces to break up word sequences that are really just combinations of words. For example, instead of transcribing 1964 as "nineteen+sixty+four", "nineteen-sixty-four", or "nineteen_sixty_four", it is best to transcribe simply as "nineteen sixty four". This principle is particularly

important for Chinese, where there is a tendency to underutilize spaces, since Chinese itself is written without spaces.

5. For most languages that use Roman characters, you can rely on capitalization to force MOR to treat words as proper nouns. To understand this, take a look at the forms in the sf.cut file at the top of the MOR directory. These various entries tell MOR how to process forms like k@l for the letter "k" or John_Paul_Jones for the famous admiral. The symbol \c indicates that a form is capitalized and the symbol \l indicates that it is lowercase.
6. Deciding how to represent compounds is a difficult matter. See the discussion in the next section.

10.5.1 Compounds and Complex Forms

The initial formulations of CHAT that were published in 1991 (1991), 1995 (1995), and 2000 (2000) specified no guidelines for the annotation of compounds. They only stipulated that compounds should be represented with a plus. When MOR saw a word with a plus, it simply tagged it as a noun compound. This was a big mistake, since many of the compounds tagged in this way were not common nouns. Instead, they included verbs, adjectives, proper nouns, idioms, greetings, onomatopoeia, and many other nonstandard forms. Unfortunately, once this genie had been let out of the bottle, it was very difficult to convince it to go back in. To solve this problem, we had to shift from blanket recognition of compounds to an exhaustive listing of the actual forms of possible compounds. The result of this shift is that we have now created many special compound files such as n+n+n.cut or v+n+v.cut. Fixing the forms in the database to correspond to this new, tighter standard was a huge job, perhaps even more tedious than that involved in removing main line morphemicization from the corpus. However, now that we have a full analysis of compounds, there is a much more accurate analysis of children's learning of these forms.

In the current system, compounds are listed in the lexical files according to both their overall part of speech (X-bar) and the parts of speech of their components. However, there are seven types of complex word combinations that should not be treated as compounds.

1. **Underscored words**. The n-dashed.cut file includes 40 forms that resemble compounds, but are best viewed as units with non-morphemic components. For example, kool_aid and band_aid are not really combinations of morphemes, although they clearly have two components. The same is true for hi_fi and coca_cola. In general, MOR and CLAN pay little attention to the underscore character, so it can be used as needed when a plus for compounding is not appropriate. The underscore mark is particularly useful for representing the combinations of words found in proper nouns such as John_Paul_Jones, Columbia_University, or The_Beauty_and_the_Beast. As long as these words are capitalized, they do not need to be included in the MOR lexicon, since all capitalized words are taken as proper nouns in English. However, these forms cannot contain pluses, since compounds are not proper nouns. And please be careful not to overuse this form.

2. **Separate words**. Many noun-noun combinations in English should just be written out as separate words. An example would be "faucet stem assembly rubber gasket holder". We don't want to write this as "Faucet_stem_assembly_rubber_gasket_holder" or "faucet_stem_assembly_rubber_gasket_holder" or even "faucet+stem+assembly+rubber+gasket+holder". It is worth noting here that German treats all such forms as single words. This means that different conventions have to be adopted for German in order to avoid the need for exhaustive listing of the infinite number of German compound nouns.
3. **Spelling sequences**. Sequences of letter names such as "O-U-T" for the spelling of "out" are transcribed with the suffix @k, as in out@k.
4. **Acronyms**. Forms such as FBI are transcribed with underscores, as in F_B_I. Presence of the initial capital letter tells MOR to treat F_B_I as a proper noun. This same format is used for non-proper abbreviations such as c_d or d_v_d.
5. **Products**. Coming up with good forms for commercial products such as Coca-Cola is tricky. Because of the need to ban the use of the dash on the main line, we have avoided the use of the dash in these names. It is clear that they should not be compounds, as in coca+cola, and compounds cannot be capitalized, so Coca+Cola is not possible. This leaves us with the option of either coca_cola or Coca_Cola. The option coca_cola seems best, since this is not really a proper noun.
6. **Interjections**. The choice between underscoring, compounding, and writing as single words is particularly tricky for interjections and set phrases. A careful study of files such as co-voc.cut, co.cut, n-dashed.cut, n-abbrev.cut, int-rhymes.cut, int.cut, inti+int+int.cut, and int+int+int.cut will show how difficult it is to apply these distinctions consistently. We continue to sharpen these distinctions, so the best way to trace these categories is to scan through the relevant files to see the principles that are being used to separate forms into these various types.
7. **Babbling and word play**. In earlier versions of CHAT and MOR, transcribers often represent sequences of babbling or word play syllables as compounds. This was done mostly because the plus provides a nice way of separating out the separate syllables in these productions. In order to make it clear that these separations are simply marked for purposes of syllabification, we now ask transcribers to use forms such as ba^ba^ga^ga@wp or choo^bung^choo^bung@o to represent these patterns.

The introduction of this more precise system for transcription of complex forms opens up additional options for programs like MLU, KWAL, FREQ, and GRASP. For MLU, compounds will be counted as single words, unless the plus sign is added to the morpheme delimiter set using the +b+ option switch. For GRASP, processing of compounds only needs to look at the part of speech of the compound as a whole, since the internal composition of the compound is not relevant to the syntax. Additionally, forms such as "faucet handle valve washer assembly" do not need to be treated as compounds, since GRASP can learn to treat sequences of nouns as complex phrases header by the final noun.

10.5.2 Lemmatization

Researchers are often interested in computing frequency profiles that are computed using lemmas or root forms, rather inflected forms. For example, they may want to treat "dropped" as an instance of the use of the lemma "drop." In order to perform these types of computations, the KWAL and FREQ programs provide a series of options that allow users to refer to various parts of complex structures in the %mor line. This system recognizes the following structures on the %mor line:

Element	Symbol	Example	Representation	Part
prefix	#	unwinding	un#v|wind-PROG	un#
stem	r	unwinding	un#v|wind-PROG	wind
suffix	-	unwinding	un#v|wind-PROG	PROG
fusion	&	unwound	un#v|wind&PAST	PAST
translation	=	gato	n|gato=cat	cat
other	o	-	-	-

To illustrate the use of these symbols, let us look at several possible commands. All of these commands take the form: freq +t%mor -t* filename.cha. However, in addition, they add the +s switches that are given in the second column. In these commands, the asterisk is used to distinguish across forms in the frequency count and the % sign is used to combine across forms.

Function	String
All stems with their parts of speech, merge the rest	+s@"r+*,|+*,o+%"
Only verbs	+s@"|+v"
All forms of the stem "go"	+s@"r+go"
The different parts of speech of the stem "go"	+s@"r+go,|+*,o+%"
The stem "go" only when it is a verb	+s@"r+go,|+v,o+%"
All stems, merge the rest	+s@"r+*,o+%"

Of these various forms, the last one given above would be the one required for conducting a frequency count based on lemmas or stems alone. Essentially CLAN breaks every element on %mor tier into its individual components and then matches either literal strings or wild cards provided by the user to each component.

10.5.3 Errors and Replacements

Transcriptions on the main line have to serve two, sometimes conflicting (Edwards, 1992), functions. On the one hand, they need to represent the form of the speech as actually produced. On the other hand, they need to provide input that can be used for morphosyntactic analysis. When words are pronounced in their standard form, these two functions are in alignment. However, when words are pronounced with phonological or morphological errors, it is important to separate out the actual production from the morphological target. This can be done through a system of main line tagging of errors. This system largely replaces the coding of errors on a separate %err line, although that form is still available, if needed. The form of the newer system is illustrated here:

```
*CHI:  him [* case] ated [: ate] [* +ed-sup] a f(l)ower and a pun [: bun].
```

For the first error, there is no need to provide a replacement, since MOR can process "him" as a pronoun. However, since the second error is not a real word form, the replacement is necessary in order to tell MOR how to process the form. The third error is just an omission of "l" from the cluster and the final error is a mispronunciation of the initial consonant. Phonological errors are not coded here, since that level of analysis is best conducted inside the Phon program (Rose et al., 2005).

10.5.4 Affixes and Control Features

To complete our tour of the MOR lexicon for English, we will take a brief look at the 0affix.cut file, as well some additional control features in the other lexical files. The responsibility of processing inflectional and derivational morphology is divided across these three files. Let us first look at a few entries in the 0affix.cut file.

1. This file begins with a list of prefixes such as "mis" and "semi" that attach either to nouns or verbs. Each prefix also has a permission feature, such as [allow mis]. This feature only comes into play when a noun or verb in n.cut or v.cut also has the feature [pre no]. For example, the verb "test" has the feature [pre no] included in order to block prefixing with "de-" to produce "detest" which is not a derivational form of "test". At the same time, we want to permit prefixing with "re-", the entry for "test" has [pre no][allow re]. Then, when the relevant rule in cr.cut sees a verb following "re-" it checks for a match in the [allow] feature and allows the attachment in this case.
2. Next we see some derivational suffixes such as diminutive –ie or agential –er. Unlike the prefixes, these suffixes often change the spelling of the stem by dropping silent e or doubling final consonants. The ar.cut file controls this process, and the [allo x] features listed there control the selection of the correct form of the suffix.
3. Each suffix is represented by a grammatical category in parentheses. These categories are taken from a typologically valid list given in the CHAT Manual.
4. Each suffix specifies the grammatical category of the form that will result after its attachment. For suffixes that change the part of speech, this is given in the scat, as in [scat adj:n]. Prefixes do not change parts of speech, so they are simply listed as [scat pfx] and use the [pcat x] feature to specify the shape of the forms to which they can attach.
5. The long list of suffixes concludes with a list of cliticized auxiliaries and reduced main verbs. These forms are represented in English as contractions. Many of these forms are multiply ambiguous and it will be the job of POST to choose the correct reading from among the various alternatives.

Outside of the 0affix.cut file, in the various other *.cut lexical files, there are several control features that specify how stems should be treated. One important set includes the [comp x+x] features for compounds. These features control how compounds will be unpacked for formatting on the %mor line. Irregular adjectives in adj-ir.cut have features specifying their degree as comparative or superlative. Irregular nouns have features controlling the use of the plural. Irregular verbs have features controlling consonant doubling [gg +] and the formation of the perfect tense.

10.5.5 Building MOR Grammars

So far, this discussion of the MOR grammar for English has avoided an examination of the ar.cut and cr.cut files. It is true that users of English MOR will seldom need to tinker with these files. However, serious students of morphosyntax need to understand how MOR and POST operate. In order to do this, they have to understand how the ar.cut and cr.cut files work. Fortunately, for English at least, these rule files are not too complex. The relative simplicity of English morphology is reflected in the fact that the ar.cut file for English has only 391 lines, whereas the same file for Spanish has 3172 lines. In English, the main patterns involve consonant doubling, silent –e, changes of y to i, and irregulars like “knives” or “leaves.” The rules use the spelling of final consonants and vowels to predict these various allomorphic variations. Variables such as $V or $C are set up at the beginning of the file to refer to vowels and consonants and then the rules use these variables to describe alternative lexical patterns and the shapes of allomorphs. For example the rule for consonant doubling takes this shape:

```
LEX-ENTRY:
LEXSURF = $O$V$C
LEXCAT = [scat v], ![tense OR past perf], ![gem no]  % to block putting
ALLO:
ALLOSURF = $O$V$C$C
ALLOCAT = LEXCAT, ADD [allo vHb]
ALLO:
ALLOSURF = LEXSURF
ALLOCAT = LEXCAT, ADD [allo vHa]
```

Here, the string $O$V$C characterizes verbs like “bat” that end with vowels followed by consonants. The first allo will produce words like “batting” or “batter” and the second will give a stem for “bats” or “bat”. A complete list of allomorphy types for English is given in the file engcats.cdc in the /docs folder in the MOR grammar.

When a user types the “mor” command to CLAN, the program loads up all the *.cut files in the lexicon and then passes each lexical form past the rules of the ar.cut file. The rules in the ar.cut file are strictly ordered. If a form matches a rule, that rule fires and the allomorphs it produces are encoded into a lexical tree based on a “trie” structure. Then MOR moves on to the next lexical form, without considering any additional rules. This means that it is important to place more specific cases before more general cases in a standard bleeding relation. There is no “feeding” relation in the ar.cut file, since each form is shipped over to the tree structure after matching.

The other “core” file in a MOR grammar is the cr.cut file that contains the rules that specify pathways through possible words. The basic idea of crules or concatenation or continuation rules is taken from Hausser’s (1999) left-associative grammar which specifies the shape of possible “continuations” as a parser moves from left to right through a word. Unlike the rules of the ar.cut file, the rules in the cr.cut file are not ordered. Instead, they work through a “feeding” relation. MOR goes through a candidate word from left to right to match up the current sequence with forms in the lexical trie TREE?? structure. When a match is made, the categories of the current form become a part of the STARTCAT. If the STARTCAT matches up with the STARTCAT of one of the rules in cr.cut, as well as satisfying some additional matching conditions specified in the rule, then that rule fires. The result of this firing is to change the shape of the STARTCAT and to then thread processing into some additional rules. For example, let us consider the processing of the verb "reconsidering." Here, the first rule to fire is the specific-vpfx-start rule which matches the fact that "re-" has the feature [scat pfx] and [pcat v]. This initial recognition of the prefix then threads into the specific-vpfx-verb rule that requires the next item have the feature [scat v]. This rule has the feature CTYPE # which serves to introduce the # sign into the final tagging to produce re#part|consider-PROG. After the verb "consider" is accepted, the RULEPACKAGE tells MOR to move on to three other rules: v-conj, n:v-deriv, and adj:v-deriv. Each of these rules can be viewed as a separate thread out of the specific-vpfx-verb rule. At this point in processing the word, the remaining orthographic material is "-ing". Looking at the 0affix.cut file, we see that "ing" has three entries: [scat part], [scat v:n], and [scat n:gerund]. One of the pathways at this point leads through the v-conj rule. Within v-conj, only the fourth clause fires, since that clause matches [scat part]. This clause can lead on to three further threads, but, since there is no further orthographic material, there is no NEXTCAT for these rules. Therefore, this thread then goes on to the end rules and outputs the first successful parse of "reconsidering." The second thread from the specific-vpfx-verb rule leads to the n:v-deriv rule. This rule accepts the reading of "ing" as [scat n:gerund] to produce the second reading of "reconsidering". Finally, MOR traces the third thread from the specific-vpfx-verb rule which leads to adj:v-deriv. This route produces no matches, so processing terminates with this result:

```
Result: re#part|consider-PROG^re#n:gerund|consider-GERUND
```

Later, POST will work to choose between these two possible readings of "reconsidering" on the basis of the syntactic context. As we noted earlier, when "reconsidering" follows an auxiliary ("is eating") or when it functions adjectivally ("an eating binge"), it is treated as a participle. However, when it appears as the head of an NP ("eating is good for you"), it is treated as a gerund. Categories and processes of this type can be modified to match up with the requirements of the GRASP program to be discussed below.

The process of building ar.cut and cr.cut files for a new language involves a slow iteration of lexicon building with rule building. During this process, and throughout work with development of MOR, it is often helpful to use MOR in its interactive mode by typing: mor +xi . When using MOR in this mode, there are several additional options that become available in the CLAN Output window. They are:

```
word - analyze this word
:q  quit- exit program
:c  print out current set of crules
:d  display application of a rules.
:l  re-load rules and lexicon files
:h  help - print this message
```

If you type in a word, such as "dog" or "perro," MOR will try to analyze it and give you its component morphemes. If all is well, you can move on the next word. If it is not, you need to change your rules or the lexicon. You can stay within CLAN and just open these using the Editor. After you save your changes, use :l to reload and retest the word again.

The problem with building up a MOR grammar one word at a time like this is that changes that favour the analysis of one word can break the analysis of other words. To make sure that this is not happening, it is important to have a collection of test words that you continually monitor using mor +xl. One approach to this is just to have a growing set of transcripts or utterances that can be analyzed. Another approach is to have a systematic target set configured not as sentences but as transcripts with one word in each sentence. An example of this approach can be found in the /verbi folder in the Italian MOR grammar. This folder has one file for each of the 106 verbal paradigms of the Berlitz Italian Verb Handbook (2005). That handbook gives the full paradigm of one "leading" verb for each conjugational type. We then typed all of the relevant forms into CHAT files. Then, as we built up the ar.cut file for Italian, we designed allo types using features that matched the numbers in the Handbook. In the end, things become a bit more complex in Spanish, Italian, and French.

1. The initial rules of the ar.cut file for these languages specify the most limited and lexically-bound patterns by listing almost the full stem, as in $Xdice for verbs like "dicere", "predicere" or "benedicere" which all behave similarly, or "nuoce" which is the only verb of its type.
2. Further in the rule list, verbs are listed through a general phonology, but often limited to the presence of a lexical tag such as [type 16] that indicates verb membership in a conjugational class.
3. Within the rule for each verb type, the grammar specifies up to 12 stem allomorph types. Some of these have the same surface phonology. However, to match up properly across the paradigm, it is important to generate this full set. Once this basic grid is determined, it is easy to add new rules for each additional conjugational type by a process of cut-and-paste followed by local modifications.
4. Where possible, the rules are left in an order that corresponds to the order of the conjugational numbers of the Berlitz Handbook. However, when this order interferes with rule bleeding, it is changed.
5. Perhaps the biggest conceptual challenge is the formulation of a good set of [allo x] tags for the paradigm. The current Italian grammar mixes together tags like [allo vv] that are defined on phonological grounds and tags like [allo vpart] that are defined on paradigmatic grounds. A more systematic analysis would probably use a somewhat larger set of tags to cover all tense-aspect-mood slots and use the phonological tags as a secondary overlay on the basic semantic tags.
6. Although verbs are the major challenge in Romance languages, it is also important to manage verbal clitics and noun and adjectives plurals. In the end, all nouns must be listed with gender information. Nouns that have both masculine and feminine forms are listed with the feature [anim yes] that allows the ar.cut file to generate both sets of allomorphs.
7. Spanish has additional complexities involving the placement of stress marks for infinitives and imperatives with suffixed clitics, such as dámelo. Italian has additional complications for forms such as "nello" and the various pronominal and clitic forms.

To begin the process, start working with the sample "minMOR" grammars available from the net. These files should allow you to build up a lexicon of uninflected stems. Try to build up separate files for each of the parts of speech in your language. As you start to feel comfortable with this, you should begin to add affixes. To do this, you need to create a lexicon file for affixes, such as affix.cut. Using the technique found in unification grammars, you want to set up categories and allos for these affixes that will allow them to match up with the right stems when the crules fire. For example, you might want to assign [scat nsfx] to the noun plural suffix in order to emphasize the fact that it should attach to nouns. And you could give the designation [allo mdim] to the masculine diminutive suffix -ito in Spanish in order to make sure that it only attaches to masculine stems and produces a masculine output.

As you progress with your work, continually check each new rule change by entering :l (colon followed by "l" for load) into the CLAN Output window and then testing some crucial words. If you have changed something in a way that produces a syntactic violation, you will learn this immediately and be able to change it back. If you find that a method fails, you should first rethink your logic. Consider these factors:

1. Arules are strictly ordered. Maybe you have placed a general case before a specific case.
2. Crules depend on direction from the RULEPACKAGES statement. Perhaps you are not reaching the rule that needs to fire.
3. There has to be a START and END rule for each part of speech. If you are getting too many entries for a word, maybe you have started it twice. Alternatively, you may have created too many allomorphs with the arules.
4. Possibly, you form is not satisfying the requirements of the end rules. If it doesn't these rules will not "let it out."
5. If you have a MATCHCAT allos statement, all allos must match. The operation DEL [allo] deletes all allos and you must add back any you want to keep.
6. Make sure that you understand the use of variable notation and pattern matching symbols for specifying the surface form in the arules.

However, sometimes it is not clear why a method is not working. In this case, you will want to check the application of the crules using the :c option in the CLAN Output window. You then need to trace through the firing of the rules. The most important information is often at the end of this output.

If the stem itself is not being recognized, you will need to also trace the operation of the arules. To do this, you should either use the +e option in standard MOR or else the :d option in interactive MOR. The latter is probably the most useful. To use this option, you should create a directory called testlex with a single file with the words you are working with. Then run: mor +xi +ltestlex

Once this runs, type :d and then :l and the output of the arules for this test lexicon will go to debug.cdc. Use your editor to open that file and try to trace what is happening there.

As you progress with the construction of rules and the enlargement of the lexicon, you can tackle whole corpora. At this point you will occasionally run the +xl analysis. Then you take the problems noted by +xl and use them as the basis for repeated testing using the +xi switch and repeated reloading of the rules as you improve them. As you build up your rule sets, you will want to annotate them fully using comments preceded by the % symbol.

10.6 Using MOR with a New Corpus

Because the English MOR grammar is stable and robust, the work of analyzing a new corpus seldom involves changes to the rules in the ar.cut or cr.cut files. However, a new English corpus is still likely to need extensive lexical clean up before it is fully recognized by MOR. The unrecognized words can be identified quickly by running this command:

```
mor +xl *.cha
```

This command will go through a collection of files and output a single file "mini lexicon" of unrecognized words. The output is given the name of the first file in the collection. After this command finishes, open up the file and you will see all the words not recognized by MOR. There are several typical reasons for a word not being recognized:

1. It is misspelled.
2. The word should be preceded by an ampersand (&) to block look up through MOR. Specifically, incomplete words should be transcribed as &text so that the ampersand character can block MOR look up. Similarly, sounds like laughing can be transcribed as &=laughs to achieve the same effect.
3. The word should have been transcribed with a special form marker, as in bobo@o or bo^bo@o for onomatopoeia. It is impossible to list all possible onomatopoeic forms in the MOR lexicon, so the @o marker solves this problem by telling MOR how to treat the form. This approach will be needed for other special forms, such as babbling, word play, and so on.
4. The word was transcribed in "eye-dialect" to represent phonological reductions. When this is done, there are two basic ways to allow MOR to achieve correct lookup. If the word can be transcribed with parentheses for the missing material, as in "(be)cause", then MOR will be happy. This method is particularly useful in Spanish and German. Alternatively, if there is a sound substitution, then you can transcribe using the [: text] replacement method, as in "pittie [: kittie]".
5. You should treat the word as a proper noun by capitalizing the first letter. This method works for many languages, but not in German where all nouns are capitalized and not in Asian languages, since those languages do not have systems for capitalization.
6. The word should be treated as a compound, as discussed in the previous section.
7. The stem is in MOR, but the inflected form is not recognized. In this case, it is possible that one of the analytic rules of MOR is not working. These problems can be reported to me at macw@cmu.edu.
8. The stem or word is missing from MOR. In that case, you can create a file called something like 0add.cut in the /lex folder of the MOR grammar. Once you have accumulated a collection of such words, you can email them to me for permanent addition to the lexicon.

Some of these forms can be corrected during the initial process of transcription by running CHECK. However, others will not be evident until you run the mor +xl command and get a list of unrecognized words. In order to correct these forms, there are basically two possible tools. The first is the KWAL program built in to CLAN. Let us say that your filename.ulx.cex list of unrecognized words has the form "cuaght" as a misspelling of "caught." Let us further imagine that you have a single collection of 80 files in one folder. To correct this error, just type this command into the Commands window:

```
kwal *.cha +scuaght
```

KWAL will then send input to your screen as it goes through the 80 files. There may be no more than one case of this misspelling in the whole collection. You will see this as the output scrolls by. If necessary, just scroll back in the CLAN Output window to find the error and then triple click to go to the spot of the error and then retype the word correctly.

For errors that are not too frequent, this method works fairly well. However, if you have made some error consistently and frequently, you may need stronger methods. Perhaps you transcribed "byebye" as "bye+bye" as many as 60 times. In this case, you could use the CHSTRING program to fix this, but a better method would involve the use of a powerful Programmer's Editor system such as BBEdit for the Mac or Epsilon for Windows. Any system you use must include an ability to process Regular Expressions (RegExp) and to operate smoothly across whole directories at a time. However, let me give a word of warning about the use of more powerful editors. When using these systems, particularly at first, you may make some mistakes. Always make sure that you keep a backup copy of your entire folder before each major replacement command that you issue.

Once you have succeeded in reducing the context of the minilex to zero, you are ready to run a final pass of MOR. After that, if there is a .db file in the MOR grammar for your language, you can run POST to disambiguate your file. After disambiguation, you should run CHECK again. There may be some errors if POST was not able to disambiguate everything. In that case, you would either need to fix MOR or else just use CLAN's disambiguate tier function (escape-2) to finish the final stages of disambiguation.

10.7 MOR for Bilingual Corpora

It is now possible to use MOR and POST to process bilingual corpora. The first application of this method has been to the transcripts collected by Virginia Yip and Stephen Matthews from Cantonese-English bilingual children in Hong Kong. In these corpora, parents, caretakers, and children often switch back and forth between the two languages. In order to tell MOR which grammar to use for which utterances, each sentence must be clearly identified for language. It turns out that this is not too difficult to do. First, by the nature of the goals of the study and the people conversing with the child, certain files are typically biased toward one language or the other. In the

YipMatthews corpus, English is the default language in folders such as SophieEng or TimEng and Cantonese is the default in folders such as SophieCan and TimCan. To mark this in the files in which Cantonese is predominant, the @Languages tier has this form:

```
    @Language:        zh, en
```

In the files in which English is predominant, on the other hand, the tier has this form:

```
    @Language:        en, zh
```

The programs then assume that, by default, each word in the transcript is in the first listed language. This default can be reversed in two ways. First, within the English files, the precode [- zh] can be placed at the beginning of utterances that are primarily in Cantonese. If single Cantonese words are used inside English utterances, they are marked with the special form marker @s. If an English word appears within a Cantonese sentence marked with the [- zh] precode, then the @s code means that the default for that sentence (Chinese) is now reversed to the other language (English). For the files that are primarily in Cantonese, the opposite pattern is used. In those files, English sentences are marked as [- en] and English words inside Cantonese are marked by @s. This form of marking preserves readability, while still making it clear to the programs which words are in which language. If it is important to have each word explicitly tagged for language, the –l switch can be used with CLAN programs such as KWAL, COMBO, or FIXIT to insert this more verbose method of language marking.

To minimize cross-language listing, it was also helpful to create easy ways of representing words that were shared between languages. This was particularly important for the names of family members or relation names. For example, the Cantonese form 姐姐 for “big sister” can be written in English as Zeze, so that this form can be processed correctly as a proper noun address term. Similarly, Cantonese has borrowed a set of English salutations such as “byebye” and “sorry” which are simply added directly to the Cantonese grammar in the co-eng.cut file.

Once these various adaptations and markings are completed, it is then possible to run MOR in two passes on the corpus. For the English corpora, the steps are:

1. Set the MOR library to English and run: mor -s”[- zh]” *.cha +1
2. Disambiguate the results with: post *.cha +1
3. Run CHECK to check for problems.
4. Set the MOR library to Cantonese and run: mor +s”[- zh]” *.cha +1
5. Disambiguate the results with: post +dcant.db *.cha +1
6. Run CHECK to check for problems.

To illustrate the result of this process, here is a representative snippet from the te951130.cha file in the /TimEng folder. Note that the default language here is English and that sentences in Cantonese are explicitly marked as [+ can].

```
*LIN:      where is grandma first, tell me ?
%mor:      adv:wh|where v|be n|grandma adv|first v|tell pro|me ?
*LIN:      well, what's this ?
%mor:      co|well pro:wh|what~v|be pro:dem|this ?
*CHI:      xxx 呢 個 唔 夠 架 . [+ can]
%mor:      unk|xxx det|ni1=this cl|go3=cl neg|m4=not adv|gau3=enough
```

sfp|gaa3=sfp . [+ can]
*LIN: 呢 個 唔 夠 . [+ can]
%mor: det|ni1=this cl|go3=cl neg|m4=not adv|gau3=enough . [+ can]
*LIN: <what does it mean> [>] ?
%mor: pro:wh|what v:aux|do pro|it v|mean ?

Currently, this type of analysis is possible whenever MOR grammars exist for both languages, as would be the case for Japanese-English, Spanish-French, Putonghua-Cantonese, or Italian-Chinese bilinguals.

10.8 POST

POST was written by Christophe Parisse of INSERM, Paris for the purpose of automatically disambiguating the output of MOR. The POST package is composed of four CLAN commands: POST, POSTTRAIN, POSTLIST, and POSTMOD. POST is the command that runs the disambiguator. POST uses a database that contains information about syntactic word order. Databases are created and maintained by POSTTRAIN and can be dumped in a text file by POSTLIST. POSTMODRULES is a utility for modifying Brill rules. In this section, we describe the use of the POST command.

In order to use POST, you must first have a database of disambiguation rules appropriate for your language. For English, this file is called eng.db. There are also POST databases now for Chinese, Japanese, Spanish, and English. As our work with POST progresses, we will make these available for additional languages. To run POST, you can use this command format :

```
post *.cha
```

This command assumes the default values of the +f, +d, and +s switches described below. The accuracy of disambiguation by POST for English will be above 95 percent. However, there will be some errors. To make the most conservative use of POST, you may wish to use the +s2 switch.

The options for POST are:

-b	do not use Brill rules (they are used by default)
+bs	use a slower but more thorough version of Brill's rules analysis.
+c	output all affixes
+cF	output the affixes listed in file F and post.db
-c	output only the affixes defined during training with POSTTRAIN (default).
-cF	omit the affixes in file F, but not the affixers defined during training with POSTTRAIN

+dF use POST database file F (default is "post.db"). This file must have been created by POSTTRAIN. If you do not use this switch, POST will try to locate a file called post.db in either the current working directory or your MOR library directory.

+e[1,2]c this option is a complement to the option +s2 and +s3 only. It allows you to change the separator used (+e1c) between the different solutions, (+e2c) before the information about the parsing process. (c can be any character). By default, the separator for +e1 is # and for +e2, the separator is /.

+f send output to file derived from input file name. If you do not use this switch, POST will create a series of output files named *.pst.

+fF send output to file F. This switch will change the extension to the output files.

-f send output to the screen

+lm reduce memory use (but longer processing time)
+lN when followed by a number the +l switch controls the number of output lines

+unk tries to process unknown words.

+sN N=0 (default) replace ambiguous %mor lines with disambiguated ones
N=1 keep ambiguous %mor lines and add disambiguated %pos lines.
N=2 output as in N=1, but with slashes marking undecidable cases.
N=3 keep ambiguous %mor lines and add %pos lines with debugging info.
N=4 inserts a %nob line before the %mor/%pos line that presents the results of the analysis without using Brill rules.
N=5 outputs results for debuging POST grammars.
N=6 complete outputs results for debuging POST grammars.
With the options +s0 and +s1, only the best candidate is outputted. With option +s2, second and following candidates may be outputted, when the disambiguation process is not able to choose between different solutions with the most probable solution displayed first. With option +s3, information about the parsing process is given in three situations: processing of unknown words (useful for checking these words quickly after the parsing process), processing of unknown rules and no correct syntactic path obtained (usually corresponds to new grammatical situations or typographic errors).

+tS include tier code S
-tS exclude tier code S
+/-t#Target_Child - select target child's tiers
+/-t@id="*|Mother|*" - select mother's tiers

10.9 POSTLIST

POSTLIST provides a list of tags used by POST. It is run on the *.db database file. The options for POSTLIST are as follows:

+dF	this gives the name of the database to be listed (default value: ‘eng.db’).
+fF	specify name of result file to be F.
+m	outputs all the matrix entries present in the database.
+r	outputs all the rules present in the database.
+rb	outputs rule dictionary for the Brill tagger.
+rn	outputs rule dictionary for the Brill tagger in numerical order.
+t	outputs the list of all tags present in the database.
+w	outputs all the word frequencies gathered in the database.
+wb	outputs word dictionary for the Brill tagger.

If none of the options is selected, then general information about the size of the database is outputted.

10.10 POSTMODRULES

This program outputs the rules used by POST for debugging rules.

10.11 POSTMORTEM

This program relies on a dictionary file called postmortem.cut to alter the part-of-speech tags in the %mor line after the final operation of MOR and POST. The use of this program is restricted to cases of extreme part-of-speech extension, such as using color names as nouns or common nouns as verbs. Here is an example of some lines in a postmortem.cut file

```
det adj v  => det n v
det adj $e  => det n $e
```

Here, the first line will change a sequence such as “the red is” from “det adj v” to “det n v”. The second line will change “det adj” to “det n” just in the case that the adjective is at the end of the sentence.

10.12 POSTTRAIN

POSTTRAIN was written by Christophe Parisse of INSERM, Paris. In order to run POST, you need to create a database file for your language. For several languages, this has already been done. If there is no POST database file for your language or your subject group, you can use the POSTTRAIN program to create this file. The default name for this file is eng.db. If you are not working with English, you should choose some other name for this file. Before running POSTTRAIN, you should take these steps:

1. You should specify a set of files that will be your POSTTRAIN training files. You may wish to start with a small set of files and then build up as you go.
2. You should verify that all of your training files pass CHECK.

3. Next, you should run MOR with the +xl option to make sure that all words are recognized.
4. You then run MOR on your training files. This will produce an ambiguous %mor line.
5. Now you open each file in the editor and use the escape-2 command to disambiguate the ambiguous %mor line.
6. Once this is done for a given file, using the Query-Replace function to rename %mor to %trn.
7. After you have created a few training files or even after you have only one file, run MOR again.
8. Now you can run POSTTRAIN with a command like this:
   ```
   posttrain +cnewdatabase.db +o0errrors.cut *.cha
   ```
9. Now, take a look at the 0errors.cut file to see if there are problems. If not, you can test out your POST file using POST. If the results seem pretty good, you can shift to eye-based evaluation of the disambiguated line, rather than using escape-2. Otherwise, stick with escape-2 and create more training data. Whenever you are happy with a disambiguated %mor line in a new training file, then you can go ahead and rename it to %trn.
10. The basic idea here is to continue to improve the accuracy of the %trn line as a way of improving the accuracy of the .db POST database file.

When developing a new POST database, you will find that eventually you need to repeatedly cycle through a standard sets of commands while making continual changes to the input data. Here is a sample sequence that uses the defaults in POST and POSTTRAIN:

```
mor *.cha +1
posttrain +c +o0errors.cut +x *.cha
post *.cha +1
trnfix *.cha
```

In these commands, the +1 must be used carefully, since it replaces the original. If a program crashes or exits while running with +1, the original can be destroyed, so make a backup of the whole directory first before running +1. TRNFIX can be used to spot mismatches between the %trn and %mor lines.

The options for POSTTRAIN are:

+a	train word frequencies even on utterances longer than length 3.
+**b**	extended learning using Brill's rules
-b	Brill's rules training only
+boF	append output of Brill rule training to file F (default: send it to screen)
+bN	parameter for Brill rules 1- means normal Brill rules are produced (default) 2- means only lexical rules are produced 3- same as +b1, but eliminates rules redundant with binary rules 4- same as +b2, but eliminates rules redundant with binary rules
+btN	threshold for Brill rules (default=2). For example, if the value is 2, a rule should

	correct 3 errors to be considered useful. To generate all possible rules, use a threshold of 0.
+c	create new POST database file with the name eng.db
+cF	create new POST database file with the name F
-c	add to an existing version of eng.db
-cF	add to an existing POST database file with the name F
+eF	the affixes and stems in file F are used for training. If this switch is not used, then, by default, all affixes are used and no stems are used. So, if you want to add stems for the training, but still keep all affixes, you will need to add all the affixes explicitly to this list.
+mN	load the disambiguation matrices into memory (about 700K) N=0 no matrix training N=2 training with matrix of size 2 N=3 training with matrix of size 3 N=4 training with matrix of size 4 (default)
+oF	append errors output to file F (default: send it to screen)
+sN	This switch has three forms N=0 default log listing mismatches between the %trn and %mor line. N=1 similar output in a format designed more for developers. N=2 complete output of all date, including both matches and mismatches
+tS	include tier code S
-tS	exclude tier code S +/-t#Target_Child - select target child's tiers +/-t@id="*|Mother|*" - select mother's tiers
+x	use syntactic category suffixes to deal with stem compounds

When using the default switch form of the error log, lines that begin with @ indicate that the %trn and %mor had different numbers of elements. Lines that do not begin with @ represent simple disagreement between the %trn and the %mor line in some category assignment. For example, if %mor has “pro:dem^pro:exist” and %trn has “co” three times. Then +s0 would yield: 3 there co (3 {1} pro:dem (2} pro:exist).

By default, POSTTRAIN uses all the affixes in the language and none of the stems. If you wish to change this behavior, you need to create a file with your grammatical names for prefixes and suffixes or stem tags. For English, we call this *traintags.cut.* However, for other languages you may want to use a different name with your +f switch. This file is used by both POSTTRAIN and POST. However, you may wish to create one file for use by POSTTRAIN and another for use by POST.

The English POST disambiguator currently achieves over 95% correct disambiguation. We have not yet computed the levels of accuracy for the other disambiguators. However, the levels may be a bit better for inflectional languages like Spanish or Italian. In order to train the POST disambiguator, we first had to create a hand-annotated training set for each language. We created this corpus through a process of bootstrapping. Here is the sequence of basic steps in training.

1. First run MOR on a small corpus and used the escape-2 hand disambiguation process to disambiguate.
2. Then rename the %mor line in the corpus to %trn.
3. Run MOR again to create a separate %mor line.
4. Run POSTTRAIN with this command: posttrain +ttraintags.cut +c +o0errors.cut +x *.cha
5. This will create a new post.db database.
6. You then need to go through the 0errors.cut file line by line to eliminate each mismatch between your %trn line and the codes of the %mor line. Mismatches arise primarily from changes made to the MOR codes in between runs of MOR.
7. Before running POST, make sure that post.db is in the right place. The default location is in the MOR library, next to ar.cut and cr.cut. However, if post.db is not there, POST will look in the working directory. So, it is best to make sure it is located in the MOR library to avoid confusion.
8. Disambiguate the MOR line with: post *.cha +1
9. Compare the results of POST with your hand disambiguation using: trnfix *.cha

In order to perform careful comparison using trnfix, you can set your *.trn.cex files into CA font and run longtier *.cha +1. This will show clearly the differences between the %trn and %mor lines. Sometimes the %trn will be at fault and sometimes %mor will be at fault. You can only fix the %trn line. To fix the %mor results, you just have to keep on compiling more training data by iterating the above process. As a rule of thumb, you eventually want to have at least 5000 utterances in your training corpus. However, a corpus with 1000 utterances will be useful initially.

During work in constructing the training corpus for POSTTRAIN, you will eventually bump into some areas of English grammar where the distinction between parts of speech is difficult to make without careful specification of detailed criteria. We can identify three areas that are particularly problematic in terms of their subsequent effects on GR (grammatical relation) identification:

1. **Adverb vs. preposition vs. particle**. The words "about", "across", "after", "away", "back", "down", "in", "off", "on", "out", "over", and "up" belong to three categories: ADVerb, PREPosition and ParTicLe. To annotate them correctly, we apply the following criteria. First, a preposition must have a prepositional object. Second, a preposition forms a constituent with its noun phrase object, and hence is more closely bound to its object than an adverb or a particle. Third, prepositional phrases can be fronted, whereas the noun phrases that happen to follow adverbs or particles cannot. Fourth, a manner adverb can be placed between the verb and a preposition, but not between a verb and a particle. To distinguish between an adverb and a particle, the meaning of the head verb is considered. If the meaning of the verb and the target word, taken together, cannot be predicted from the meanings of the verb and the target word separately, then the target word is a particle. In all other cases it is an adverb.
2. **Verb vs. auxiliary**. Distinguishing between Verb and AUXiliary is especially tricky for the verbs "be", "do" and "have". The following tests can be applied. First, if the target word is accompanied by a nonfinite verb in the same clause, it

is an auxiliary, as in “I have had enough” or “I do not like eggs”. Another test that works for these examples is fronting. In interrogative sentences, the auxiliary is moved to the beginning of the clause, as in “Have I had enough?” and “Do I like eggs?” whereas main verbs do not move. In verb-participle constructions headed by the verb “be”, if the participle is in the progressive tense (“John is smiling”), then the head verb is labeled as an AUXiliary, otherwise it is a Verb (“John is happy”).

3. **Communicator vs. Interjection vs. Locative adverbs**. COmmunicators can be hard to distinguish from interjections, and locative adverbs, especially at the beginning of a sentence. Consider a sentence such as “There you are” where “there” could be interpreted as either specifying a location or as providing an attentional focus, much like French voilà. The convention we have adopted is that CO must modify an entire sentence, so if a word appears by itself, it cannot be a CO. For example, utterances that begin with “here” or “there” without a following break are labelled as ADVerb. However, if these words appear at the beginning of a sentence and are followed by a break or pause, then they are labelled CO. Additionally, for lack of a better label, in here/there you are/go, here or there are labelled CO. Interjections, such as “oh+my+goodness” are often transcribed at the beginning of sentences as if they behaved like communicators. However, they might better be considered as sentence fragments in their own right.

10.13 POSTMOD

This tool enables you to modify the Brill rules of a database. There are these options:

+dF	use POST database file F (default is eng.db).
+rF	specify name of file (F) containing actions that modify rules.
+c	force creation of Brill's rules.
+lm	reduce memory use (but increase processing time).

11 GRASP – Syntactic Dependency Analysis

This chapter, written by Eric Davis, Shuly Wintner, Brian MacWhinney, Alon Lavie, and Kenji Sagae, describes a system for coding syntactic dependencies in the English CHILDES corpora. This system uses the GRASP program in CLAN to process the information on the %mor line to automatically create a dependency analysis. Here we describe first the annotation system and then the use of the program.

11.1 Grammatical Relations

GRASP describes grammatical relations in terms of dependencies. Following the formalization of Mel'cuk (2006), dependencies involve both valency relations and attachment relations. Attachment relations specify the relation between a head and a dependent. Valency relations specify the relation between a predicate and one of its several possible arguments. Valency relations open up slots for arguments. In English, modifiers (adjectives, determiners, quantifiers) are predicates whose arguments are the following nouns. In this type of dependency organization the argument becomes the head. However, in other grammatical relations, the predicate or governor is the head and the resultant phrase takes on its functions from the predicate. Examples of predicate-head GRs include the attachment of thematic roles to verbs and the attachment of adjuncts to their heads. In the notation we use for GRs, the relations between heads and dependents are made explicit. However, valency or government relations are only coded implicitly. Therefore, for each of the GRs listed, we will state which element serves as the head and whether that element is also the predicate or governor.

The following is a comprehensive list of the grammatical relations in the CHILDES GR annotation scheme. Example GRs as well as other relevant GR to that particular GR are provided. In this annotation scheme, C refers to clausal and X refers to non-finite clausal. This list is divided into relations in which the predicate becomes the head and relations in which the argument becomes the head. In the examples, the dependent is marked in italics.

Predicate-head relations. First, we list the relations in which the dependent attaches to a governing head.

1. SUBJect identifies the subject of clause, when the subject itself is not a clause. Typically, the head is the main verb and the dependent is a nominal. Ex: *You* eat with your spoon.
2. ClausalSUBJect = CSUBJ identifies the finite clausal subject of another clause. The head is the main verb, and the dependent is the main verb of the clausal subject. Ex: That Eric *cried* moved Bush.
3. XSUBJect identifies the non-finite clausal non-finite subject of another clause. The head is the main verb, and the dependent is the main verb of the clausal subject. Ex: *Eating* vegetables is important.
4. OBJect identifies the first object of a verb. The head is the main verb, and the dependent is a nominal or a noun that is the head of a nominal phrase. A clausal

complement relation should be denoted by COMP or XCOMP (depending on whether the clausal complement is finite or non-finite, see below), not OBJ or OBJ2. Ex: You read the *book*.

5. OBJect2 = OBJ2 identifies the second object of a ditransitive verb, when not introduced by a preposition. The head is a ditransitive verb, and the dependent is a noun (or other nominal). The dependent must be the head of a required non-clausal and nonprepositional complement of a verb (head of OBJ2) that is also the head of an OBJ relation. A second complement that has a preposition as its head should be denoted by IOBJ, not OBJ2. Ex: He gave *you* your telephone.
6. IndirectOBJect = IOBJ identifies an (required) object complement introduced by a preposition. When a prepositional phrase appears as the required complement of a verb, it is the dependent in an IOBJ relation, not a JCT (adjunct) relation. The head is the main verb, and the dependent is a preposition (not the complement of the preposition, see POBJ below). Ex: Mary gave a book to *John*.
7. LOCative identifies the relation between a verb and a required location. Locations are required for verbs such as *put* or *live*. LOC takes the place of JCT in such cases when the PP is required by the verb. This is especially relevant for *here*, *there*, and *back*, which would otherwise be labeled JCT for other verbs. Ex: Put the toys in the *box*.
8. COMPlement identifies a finite clausal complement of a verb. The head is the main verb of the matrix clause, and the dependent is the main verb of the clausal complement. Ex: I think that *was* Fraser.
9. XCOMPlement identifies a non-finite clausal complement of a verb. The head is the main verb of the matrix clause, and the dependent is the main verb of the clausal complement. The XCOMP relation is only used for non-finite clausal complements, not predicate nominals or predicate adjectives (see PRED). Ex: You’re going to *stand* on my toe. I told you to *go*. Eve, you stop *throwing* the blocks.
10. PREDicate identifies a predicate nominal, predicate adjective, or a prepositional complement of verbs such as *be* and *become*. The head is the verb. PRED should not be confused with XCOMP, which identifies a non-finite complement of a verb (some syntactic formalisms group PRED and XCOMP in a single category). Ex: I’m not *sure*. He is a *doctor*. He is *in* Chicago.
11. ClausalPREDicate = CPRED identifies a finite clausal predicate that identifies the status of the subject of verbs such as *be* and *become*. The head is the main verb of the matrix clause, not its subject. The dependent is the verb of the predicate clause Ex: This is how I *drink* my coffee.
12. XPREDicate identifies a non-finite clausal predicate of the subject of verbs such as *be* and *become*. The head is the main verb (of the matrix clause), not its subject. Ex: My goal is to *win* the competition.
13. PrepositionalOBJect = POBJ is the relation between a preposition and its object. The head is a preposition, and the dependent is typically a noun. The traditional treatment of the prepositional phrase views the object of the preposition as the head of the prepositional phrase. However, we are here treating the preposition as the head, since the prepositional phrase then participates in a further JCT relation

to a head verb or a NJCT relation to a head noun. Ex: You want to sit on the *stool*?

Argument-head relations: Relations in which dependents serve as governors include relations of adjunction and modification.

1. adJunCT = JCT identifies an adjunct that modifies a verb, adjective, or adverb. The adjunct is the governor, since it opens up a valency slot for something to attach to. The head of JCT is the verb, adjective or adverb to which the JCT attaches as a dependent. The dependent is typically an adverb, a preposition (in the case of phrasal adjuncts headed by a preposition, such as a prepositional phrase). Intransitive prepositions may be treated as adverbs, in which case the JCT relation applies. Adjuncts are optional, and carry meaning on their own (and do not change the basic meaning of their JCT heads). Verbs requiring a complement describing location may be treated as prepositional objects, in which case the IOBJ relation applies (see above). Ex: That's *much* better. He ran *with* a limp.
2. ClausaladJunCT = CJCT identifies a finite clause that adjoins to a verb, adjective, or adverb head. The dependent is typically the main verb of a subordinate clause. Ex: We can't find it, because it *is* gone.
3. TAG is the relation between the finite verb of a tag question and the root verb of the main clause. Ex: You know how to count, *don't* you?
4. XadJunCT = XJCT identifies a non-finite clause that adjoins to a verb, adjective, or adverb. The dependent is typically the main verb of a non-finite subordinate clause. Ex: She's outside *sleeping* in the carriage.
5. Nominal adJunCT = NJCT identifies the head of a complex NP with a prepositional phrase attached as an adjunct of a noun. Ex: The man *with* an umbrella arrived late.
6. MODifier identifies a non-clausal nominal modifier or complement. The head is a noun, and the dependent is typically an adjective, noun or preposition. Ex: Would you like *grape* juice? That's a *nice* box.
7. ClausalMODifier = CMOD identifies a finite clause that is a nominal modifier (such as a relative clause) or complement. The head is a noun, and the dependent is typically a finite verb. Ex: Here are the grapes I *found*.
8. XMODifier identifies a non-finite clause that is a nominal modifier (such as a relative clause) or complement. The head is a noun, and the dependent is typically a non-finite verb. Ex: It's time to *take* a nap.
9. DETerminer identifies a determiner of a noun. Determiners include *the*, *a*, as well as (adjectival) possessives pronouns (*my*, *your*, etc) and demonstratives (*this*, *those*, etc), but not quantifiers (*all*, *some*, *any*, etc; see QUANT below). Typically, the head is a noun and the dependent/governor is a determiner. In cases where a word that is usually a determiner does not have a head, there is no DET relation. Ex: I want *that* cookie.
10. QUANTifier identifies a nominal quantifier, such as three, many, and some. Typically, the head is a noun, and the dependent is a quantifier. In cases where a quantifier has no head, there is no QUANT relation. In English, the MOD, DET,

and QUANT relations have largely the same syntax. However, within the noun phrase, we occasionally see that they are ordered as QUANT+DET+MOD+N. Ex: I'll take *three* bananas.

11. PostQuantifier = PQ is the relation between a postquantifier and the preceding head nominal. Ex: We *both* arrived late.
12. AUXiliary identifies an auxiliary of a verb, or a modal. The head is a verb, and the dependent is an auxiliary (such as be or have) or a modal (such as *can* or *should*). Ex: *Can* you do it?
13. NEGation identifies verbal negation. When the word *not* (contracted or not) follows an auxiliary or modal (or sometimes a verb), it is the dependent in a NEG relation (not JCT), where the auxiliary, modal or verb (in the absence of an auxiliary or modal) is the head. Ex: Mommy will *not* read it.
14. INFinitive identifies an infinitival particle (to). The head is a verb, and the dependent is always to. Ex: He's going *to* drink the coffee.
15. SeRLial identifies serial verbs such as go play and come see. In English, such verb sequences start with either *come* or *go*. The initial verb is the dependent, and the verb next to the inital verb, e.g., play and see in the previous example, is the head (the adjacent verb is typically the root of the sentence). Ex: *Come* see if we can find it. *Go* play with your toys over there.
16. ComPlementiZeR = CPZR identifies the relation between a complementizer (*that*, *which*) or a subordinate conjunction and the verb to which it attaches. After this attachment, the verbal head acts as the dependent in a CJCT relation involving the embedded clause and its matrix clause (the verb is higher in the dependency tree than the complementizer). Ex: Wait *until* the noodles are cool.

Root linkage. There is also a set of relations in which the dependent is a sentential modifier. These could be viewed as depending on either the root or the left wall. Somewhat arbitrarily, we code them as linking to the root.

1. COMmunicator identifies a communicator (such as *hey*, *okay*, etc). Because communicators are typically global in a given sentence, the head of COM is typically the root. The dependent is a communicator. COM items often appear either at the very beginning or very end of a clause or sentence. Ex: *Yes*, you got a fly. You need more paper, *right*?
2. VOCative identifies a vocative. As with COM, the head is the root of the sentence. The dependent is a vocative. Ex: Some more cookies, *Eve*?
3. TOPicalization identifies an object or a predicate nominal that has been topicalized. The head is the ROOT of the sentence, and the dependent is the topicalized item. Ex: *Tapioca*, there is no tapioca. In other languages, topics may appear without repetition, as in *Tapioca, there is no.*
4. INCROOT identifies a word that serves as the root of an utterance, because the usual root forms (verbs in English) are missing. This form could be a single word by itself (adverb, communicator, noun, adjective) or a word with additional modifiers, such as the noun *dog* in *the big dog*, when it occurs by itself without a verb. It may appear that there could be more than one of these in an utterance, as in *well, sure*. However, in this case, *well* should be marked as a CO that is

dependent on *sure*.

Cosmetic relations. There are several relations that are just used to keep other relations straight:

1. PUNCTuation is the relation between the final punctuation mark and the root.
2. RightDislocationPunctuation = RDP is the relation between the marker of right dislocation or topicalization and the root. This is just a notational convention.
3. VocativePunctuation = VOCP is the relation between the mark of the vocative and the root.
4. ROOT This is the relation between the topmost word in a sentence (the root of the dependency tree) and the LeftWall. The topmost word in a sentence is the word that is the head of one or more relations, but is not the dependent in any relation with other words (except for the LeftWall).

Series relations. Some additional relations involve processes of listing, coordination, and classification. In these, the final element is the head and the initial elements all depend on the final head. In English, this extends the idea that the last element in a compound is the head.

1. NAME identifies a string of proper names such as Eric Davis and New York Central Library. The initial name is the dependent, and the adjacent name is the head. The adjacent name is the dependent of the ROOT. Ex: My name is *Tom* Jones.
2. DATE identifies a date with month and year, month and day, or month, day, and year. Examples include: October 7, 1980 and July 26. For consistency, we regard the final element in these various forms as the head. Ex: *October seventh nineteen* ninety.
3. ENUMeration involves a relation between elements in a series without any coordination based on a conjunction (*and, but, or*). The series can contain letters, numbers, and nominals. The head is the last item in the series, and all the other items in the enumeration depend on this last word. Ex: *one, two, three*, four.
4. CONJ involves a relation between a coordinating conjunction and one or more preceding items. For example, in the phrase *I walk, jump, and run*, the items *walk* and *jump* are both dependents and the item *and* is the governing head. The resultant phrase *walk, jump and* is then further linked by the COORD relation to the final element *ran*. Ex: I *walk, jump,* and run.
5. COORD involves an attachment of a conjoined phrase such as *walk and* with a final coordinated element, which then serves as the head of the entire conjoined phrase. (In the current training corpus, there is a single COORD relation with the conjunction as the head. However, that analysis is incorrect and will be changed soon.) Ex: Tom, Bill, *and* Frank arrived on the late train.

Bivalency. The above GRs describe dependencies in terms of the direction of the major valency relations. However, many of these relations have a secondary bivalent nature.

For example, in the relations of thematic roles with the verb, it is also true that nominals are "looking for" roles in predications. Within the noun phrase, common nouns are looking for completion with either articles or plurality. Also, the attachment of auxiliaries to non-finite verbs serves to complete their finite marking. We can think of these additional valency relations as secondary relations. In all of these cases, valency works within the overall framework of dependency. Because GRASP relations are unidirectional, bivalency cannot be represented in GRASP.

11.2 Ellision Relations

In addition to the basic GRs, there is this additional set of GRs used for marking elided elements.

1. AUX-ROOT identifies an auxiliary of a verb, or a modal with an elided main verb. Typically, the AUX-ROOT is the head of the entire utterance. Ex: Yes, I can xxx.
2. AUX-COMP identifies an auxiliary of a verb, or a modal with an elided complement. The head is a verb, and the dependent is an auxiliary (such as *be* or *have*) or a modal (such as *can* or *should*). Ex: I wish you would xxx.
3. AUX-COORD identifies an auxiliary of a verb, or a modal with an elided coordinated item. The head is the coordinator or verb, and the dependent is an auxiliary (such as *be* or *have*) or a modal (such as *can* or *should*). Ex: xxx and he will.
4. DET-OBJ identifies a determiner of a noun with an elided object. Determiners include the, a, as well as (adjectival) possessives pronouns (my, your, etc). Typically, the DET-OBJ depends on a verb. This GR mainly results from an interruption by another speaker (where +... indicates a trailing off). Ex: have a + …
5. DET-POBJ identifies a determiner of a noun with an elided prepositional object. Determiners include the, a, as well as (adjectival) possessives pronouns (my, your, etc). The DET-POBJ depends on a preposition or adverb. This GR mainly results from an interruption by another speaker. Ex: climb up the +…
6. DET-COORD identifies a determiner of a noun with an elided coordinated object. Determiners include the, a, as well as (adjectival) possessives pronouns (*my*, *your*, etc). Typically, the DET-COORD depends on a coordinator, but it can also depend on a verb. This GR mainly results from an interruption by another speaker. Ex: and a +…
7. DET-JCT identifies a determiner of a noun with an elided adjunct. Determiners include the, a, as well as (adjectival) possessives pronouns (*my*, *your*, etc). The DET-JCT depends on a verb (usually the main verb). Ex: Eve sit down another + …
8. INF-XCOMP identifies an infinitival particle (to) with an elided verbal complement. The head is a verb, and the dependent is always to. Ex: Yes, I want to, too.
9. INF-XMOD identifies an infinitival particle (to) with an elided verbal modifier. The head is usually a nominal, and the dependent is always to. Ex: time to +…

10. QUANT-OBJ identifies a nominal quantifier, such as *three*, *many*, and *some* with an elided object. Typically, the head is the elided noun, and the QUANT-OBJ depends on the main verb. Ex: You've just had some xxx.
11. QUANT-POBJ identifies a nominal quantifier, such as *three, many*, and *some* with an elided prepositional object. Typically, the head is the elided prepositional object, and the QUANT-POBJ depends on the preposition. Ex: There's not room for both xxx.
12. QUANT-PRED identifies a nominal quantifier, such as three, many, and some with an elided predicate. Typically, the head is the elided predicate, and the QUANT-PRED depends on the main verb. Ex: That's too much.
13. QUANT-COORD identifies a nominal quantifier, such as *three*, *many*, and *some* with an elided coordinated object (usually nominal). Typically, the head is the coordinated item, and the QUANT-COORD depends on the coordinator. Ex: You had more and more cookies.

11.3 GRs for Chinese

The GRs needed for Chinese are not too very different from those needed for English.
The major differences involve these areas:

1. Chinese uses all the basic GRs that are also found in English with these exceptions: TAG, DET, and INF.
2. Also, Chinese does not have a finite, non-finite distinction on verbs. Somewhat arbitrarily, this makes the "X" relations of English irrelevant and only CSUBJ, COMP, CMOD, CPRED, and CJCT are needed. Also, the main verb is the head of the clause, not the auxiliary.
3. Chinese makes extensive use of topics and sentence final particles. In the TOP relation, the head/root can be a nominal or adjective, as well as a verb.
4. Chinese has a variety of verb-verb constructions, beyond simple serial verbs. For Chinese, the head of SRL is the first verb, not the second.
5. The Chinese possessive construction has the head in final position.
   Also, Chinese can use classifier phrases such as *yi1 bian4* as JCT
6. Chinese often combines clauses without using any conjunction to mark subordination or coordination. In this case, it is best to transcribe the two clauses as separate sentences. To mark the missing subordination relation, just add this postcode to the end of the first sentence: [+ sub]. This mark does not necessarily imply which clause is subordinate; it just notes that the two clauses are related, although the relation is not marked with a conjunction.
7. The CJCT relation can also extend to mei2 you3 because they are both words
   neg|mei2=not v|you3=have v|jiang3=speak v:resc|wan2=finish .
   1|2|NEG 2|3|CJCT 3|0|ROOT 4|3|VR 5|3|PUNCT
8. The CJCT relation is also used in the many cases where there are no subordinating conjunctions, as in this example:
   n|qing1wa1 adv|yi1 v|kan4 adv|jiu4 neg|mei2 sfp|le
   1|3|SUBJ 2|3|JCT 3|5|CJCT 4|5|JCT 5|0|INCROOT 6|5|SFP
9. Clefts can be coded using the CPRED relations as in this example:
   co|lai2 n:relat|jie3&DIM adv|jiu4 v:cop|shi4 adv|zhe4yang4 v|jiang3 sfp|de

1|4|COM 2|4|SUBJ 3|4|JCT 4|0|ROOT 5|6|JCT 6|4|CPRED 7|4|SFP 8|4|PUNCT

10. The JCT relation for Chinese extends also to the complements of directional verbs, as in this example:
    v|pao3=run v:dirc|jin4=enter n|ping2=bottle post|li3=inside
    1|0|ROOT 2|1|VD 3|4|POSTO 4|1|JCT
    Note that the JCT is attached to the second of the two serial verbs

The additional relations needed for Chinese are:

1. POSSession = POSS is the relation that holds between the linker "de" and the preceding possessor noun or pronoun which then functions as the head for further attachment to the thing possessed through the MOD relation.
2. Chinese has no articles and uses classifiers (CLASS) to mark quantification.
3. VR is the relation between the initial verb as head and the following resultative verb as dependent.
4. VD is the relations between the initial verb in a serial verb construction as head and the following directional verb as dependent.
5. SFP is a relation between the sentence final particle and the root.
6. PTOP is a postposed topic, as in this example:

pro|zhe4=this class|ge4 v:cop|shi4=is pro:wh|shen2me=what pro|zhe4=this class|ge4
1|2|DET 2|3|SUBJ 3|0|ROOT 4|3|OBJ 5|6|DET 6|3|SUBJ

7. PostpositionalObject = POSTO is the relation between a postposition, which is the head and the dependent noun. This relation is marked on the noun, as in this example:

   ta pao dao jiaoshu limian qu 1|2|SUBJ 2|0|ROOT 3|2|JCT 4|5|POSTO 5|3|JCT 6|2|SRL
8. PrepositionalObject = PREPO is the relation between a preposition, which is the head and the following dependent noun, as in this example, where the relation is coded on the noun:
   ta shi cong Beijing lai de 1|2|SUBJ 2|0|ROOT 3|5|JCT 4|3|PREPO 5|2|CJCT 6|5|CPZR
   This relation can also hold between a preposition and a postposition, as in this example:
   v|yang3=maintain prep|zai4=at pro:wh|shen2me=what post|li3mian4=inside ?
   1|0|ROOT 2|1|JCT 3|4|POSTO 4|2|PREPO 5|1|PUNCT

11.1 GRs for Japanese

The GRs needed for Japanes are more differentiated than those needed for English. The major differences involve these areas:

1. Japanese is left-branched with the head following the modifier, adjunct or complement.
2. Japanese uses mostly optional case particles to identify case relations. This makes a range of argument relations necessary (besides SUBJ and OBJ, ORIG, DIREC, INSTR, SYM, and LOC are used). The particles themselves are coded

as CASP. The noun is coded as the head of the case particle.
Ken ga Tookyoo kara kita
Ken SUBJ Tokyo ORIG come-PAST “Ken came from Tokyo”
1|5|SUBJ 2|1|CASP 3|5|ORIG 4|3|CASP 5|0|ROOT

3. The root can be a tense-bearing element like a verb, a verbal adjective (ROOT), a copula (COPROOT) or a noun (PREDROOT).
4. The root can be also a quotative marker (QUOTROOT), a conjunctive particle (CPZRROOT) or a topic (TOPROOT). Note that these different types of roots are fully grammatical and not elliptic fragments.
   iku kara. go-PRES because “because (I) will go” 1|2|COMP 2|0|CPZRROOT
5. Japanese expresses topic relations (TOP); the topic particle is coded as PTL.
6. Like Chinese, Japanese has no articles and uses classifiers and counters to mark quantification.
   sankurambo sanko tabeta. 3-pieces eat-PAST “he ate 3 cherries”
   1|3| OBJ 2|1|QUANT 3|0|ROOT
7. Japanese makes extensive use of focus particles (FOC), sentence final particles (SFP) and sentence modifiers (SMDR).

<table>
<tr><td>Root</td><td>ROOT</td><td>verbal ROOT; relation between verb and left wall: v, adj, subsidiary verb (tense bearing element)
<i>taberu.</i>
1|0|ROOT</td></tr>
<tr><td></td><td>COPROOT</td><td>COPula ROOT; copula with noun, adjectival noun, or sentence nominalizer (<i>no da</i>)
<i>koko da.</i>
1|2|PRED 2|0|COPROOT</td></tr>
<tr><td></td><td>PREDROOT</td><td>nominal ROOT (without copula); includes adv, co, and quant in root position.
<i>koko.</i>
1|0|PREDROOT</td></tr>
<tr><td></td><td>CPREDROOT</td><td>nominal ROOT with a sentence nominalizer in root position (ptl:snr|no)
<i>uma no chiisai no.</i>
1|3|MOD 2|1|CASP 3|4|CMOD 4|0|CPREDROOT</td></tr>
<tr><td>Topic</td><td>TOP</td><td>TOPicalization, (for convenience the root of the sentence is considered to be the head)
<i>kore wa yomenai.</i>
1|3|TOP 2|1|TOPP 3|0|ROOT</td></tr>
<tr><td></td><td>CTOP</td><td>finite Clausal TOPic (head of the clause is ptl:snr|no)
<i>iku no wa ii kedo [...]</i>
1|2|CMOD 2|4|CTOP 3|2|TOPP 4|5|COMP 5|6|CPZR</td></tr>
<tr><td></td><td>FOC</td><td>FOCus (followed by ptl:foc; <i>mo, shika, bakari, hodo</i> etc.)
<i>kore mo yonda.</i>
1|3|FOC 2|1|FOCP 3|0|ROOT</td></tr>
<tr><td>Arguments</td><td>SUBJ</td><td>nonclausal SUBject</td></tr>
</table>

		Jon ga tabeta. 1|3|SUBJ 2|1|CASP 3|0|ROOT
	CSUBJ	finite Clausal SUBJect (head of the clause is ptl:snr) *taberu no ga ii.* 1|2|CMOD 2|3|CSUBJ 3|0|ROOT
	OBJ	accusative OBJect *hon o yonda.* 1|3|OBJ 2|1|CASP 3|0|ROOT
	COBJ	finite Clausal accusative OBJect *taberu no o yameta.* 1|2|CMOD 2|4|COBJ 3|2|CASP
	ORIG	ORIGinalis (incl. temporalis & passive agent) *gakkoo kara kaetta.* 1|3|ORIG 2|1|CASP 3|0|ROOT
	DIREC	DIRECtionalis (incl. temporalis, motive, benefactive) *gakkoo ni iku.* 1|3|DIREC 2|1|CASP 3|0|ROOT
	INSTR	INSTRumentalis (tool; material) *fooku de taberu.* 1|3|INSTR 2|1|CASP 3|0|ROOT
	CINSTR	finite Clausal INSTRumentalis *ochita no de taberu.* 1|2|CMOD 2|3|CINSTR 3|2|CASP 4|0|ROOT
	SYM	SYMmetry *Papa to asonda.* 1|3|SYM 2|1|CASP 3|0|ROOT
	CSYM	finite Clausal SYMmetry *moratta no to asobu.* 1|2|CMOD 2|4|CSYM 3|2|CASP 4|0|ROOT
	LOC	LOCative *uchi de tabeta.* 1|3|LOC 2|1|CASP 3|0|ROOT
Clause conjunction	CPZR	ComPlementiZeR (subordinating conjunctive particle; ptl:conj|) *osoi kara kaeru.* 1|2|COMP 2|3|CPZR 3|0|ROOT
	ZCPZR	Zero-ComPlementiZeR (sentence introducing conjunction); head is always the root *dakara kaeru.* 1|2|ZCPZR 2|0|ROOT
	COMP	finite clausal verb COMPlement (before ptl:conj| and quot|to) *osoi kara kaeru.* 1|2|COMP 2|3|CPZR 3|0|ROOT
	QUOT	QUOTative after nominal or verbal phrase *Juria to iimasu.* 13COMP 21QUOT 30ROOT

	ZQUOT	Zero-QUOTative (sentence introducing quotative marker) *tte iu ka [...]* 1|2|ZQUOT 2|3|COMP 3|4|CPZR
Nominal head	MOD	nonclausal MODifier (of a nominal) *Papa no kutsu ga atta.* 1|3|MOD 2|1|CASP 3|5|SUBJ 4|3|CASP 5|0|ROOT
	MOD-SUBJ	modifier in subject position with head-noun elided *Papa no ga atta.* 1|4|MOD-SUBJ 2|1|CASP 3|4|CASP 4|0|ROOT
	MOD-OBJ	modifier in object position with head-noun elided *Papa no o mita.* 1|3|MOD-OBJ 2|1|CASP 3|4|CASP 4|0|ROOT
	MOD-INSTR	modifier in instrumentalis position with head-noun elided *Papa no de asonda.* 1|3|MOD-INSTR 2|1|CASP 3|4|CASP 4|0|ROOT
	MOD-PRED	modifier in predicate position with head-noun elided *kore wa Papa no da.* 1|5|TOP 2|1|TOPP 3|5|MOD-PRED 4|3|CASP 5|0|COPROOT
	MOD-TOP	modifier in topic position with head-noun elided *Papa no wa agenai.* 1|4|MOD-TOP 2|1|CASP 3|2|TOPP 4|0|ROOT
	CMOD	finite Clausal MODifier of a nominal; the dependent is a finite verb, adjective or adj noun with copula *akai kuruma o mita.* 1|2|CMOD 2|4|OBJ 3|2|CASP 4|0|ROOT
	XMOD	nonfinite clausal MODifier of a nominal (adn|) *kore to onaji mono ga [...]* 1|3|SYM 2|1|CASP 3|4|XMOD 4|6|SUBJ 5|4|CASP
	COORD	COORDination, second noun is the head; (ptl:coo|) *inu to neko o katte iru.* 1|3|COORD 2|1|COOP 3|5|OBJ 4|3|CASP 5|6|XJCT 6|0|ROOT
Verbal head	JCT	adverbial adJunCT to a verbal head; (adv|) *yukkuri shabetta.* 1|2|JCT 2|0|ROOT
	XJCT	nonfinite clause as adJunCT (*tabe-reba, -tara, -te, -cha, -tari; oishi-ku; shizuka ni*) *tsunagereba ii.* 1|2|XJCT 2|0|ROOT
NP	PRED	nominal PREDicate before copula or QUOT *tabeta hito da.* 1|2|CMOD 2|3|PRED 3|0|COPROOT
	CPRED	finite Clausal PREDicate before copula (*no da*) *taberu no da.* 1|2|CMOD 2|3|CPRED 3|0|COPROOT
	CASP	CASe Particles (ptl:case; *ga, o, kara, ni, de, to, no*) *hon o yonda.* 1|3|OBJ 2|1|CASP 3|0|ROOT
	TOPP	TOPic Particle (ptl:top) *kore wa yomenai.*

		1|3|TOP 2|1|TOPP 3|0|ROOT
	FOCP	FOCus Particle (ptl:foc; *mo, shika, bakari, hodo* etc.) *kore mo yonda.* 1|3|OBJ 2|1|FOCP 3|0|ROOT
	COOP	COOrdination Particles *inu to neko o [...]* 1|3|COORD 2|1|COOP 3|5|OBJ 4|3|CASP
	QUANT	QUANTifier (incl. classifiers and counters) *banana sambon tabeta.* 1|3|OBJ 2|1|QUANT 3|0|ROOT
	ENUM	ENUMeration, without coordinating particle *ichi ni sanko da.* 1|2|ENUM 2|3|ENUM 3|4|JCT 4|0|ROOT
	NAME	string of proper NAMEs, second name is the head *Kameda Taishoo ga kita.* 1|2|NAME 2|4|SUBJ 3|2|CASP 4|0|ROOT
	DATE	string of DATEs, last element (day) is the head *rokugatsu gonichi ni kita.* 1|2|DATE 2|4|LOC 3|2|CASP 4|0|ROOT
Others	SMDR	sentence final Sentence MoDifieR (smod| *mitai, jan, rashii* etc); for convenience, the tense bearing verb is considered to be the head *kaetta mitai.* 1|0|ROOT 2|1|SMDR
	SFP	Sentence Final Particle (including the use after arguments) *kuru ne.* 1|0|ROOT 2|1|SFP
	COM	COMmunicator; (co:i| co:g|) including isolated final particles, sentence modalizers and onomatopoeias; head is always set to 0 *anoo tabeta.* 1|0|COM 2|0|ROOT
	VOC	VOCative ; head is always set to 0 *Taishoo ‡ aka.* 1|0|VOC 2|3|VOCP 3|0|PREDROOT
Punctuation	PUNCT	sentence boundary (sentence ends; .!? etc.); the root is the head *iku .* 1|0|ROOT 2|1|PUNCT
	RDP	Right Dislocation boundary (tag|„); dislocation follows; the root is the head *mita „ fuusen ?* 1|0|ROOT 2|1|RDP 3|1|SUBJ 4|1|PUNCT
	VOCP	VOCative marker (tag|‡); head is the preceding vocative *Taishoo ‡ mite !* 1|0|VOC 2|3|VOCP 3|0|ROOT 4|3|PUNCT

11.2 MEGRASP

The MEGRASP program uses a training corpus that is distributed in the English MOR grammar to train a statistical parser. This corpus contains a hand-annotated %grt tier to control training. The output of the parser is the megrasp.mod file that is then used to create a new %xgra coding tier for the English CHILDES corpora. The program uses these options:

-e : evaluate accuracy (input file must contain gold standard GRs)
-t : training mode (parser runs in parse mode by default)
-iN: number of iterations for training ME model (default: 300)
-cN: inequality parameter for training ME model (default: 0.5).
+fS: send output to file (program will derive filename)
+re: run program recursively on all sub-directories.

12 Utility Commands

The various utility commands are used primarily for fixing and reformatting older files to bring them into accord with the current CHAT format or for reformatting data for use with other programs.

Command	Page	Function
CHAT2CA	186	Convert CA/CHAT to purer CA for display only
CHAT2ELAN	187	Convert CHAT to ELAN format
CHAT2XMAR	187	Convert CHAT to EXMARaLDA format
CHSTRING	187	Changes words and characters in CHAT files.
COMPOUND	189	Converts word pairs to compounds
COMBTIER	190	Combines extra commentary lines.
CP2UTF	190	Converts ASCII files to Unicode files.
DATACLEAN	190	Updates the format of older CHAT files.
DATES	190	Uses the date and birthdate of the child to compute age.
DELIM	191	Inserts periods when final delimiters are missing.
ELAN2CHAT	191	Converts ELAN files to CHAT
FIXBULLETS	191	Repairs bullets and reformats old style bullets
FIXIT	192	Breaks up tiers with multiple utterances.
FIXMP3S	192	Fixes bullets to MP3 media in older CHAT files
FLO	192	Reformats the file in simplified form.
INDENT	192	Aligns the overlap marks in CA files.
INSERT	192	Inserts @ID fields.
LIPP2CHAT	192	Converts LIPP files to CHAT
LONGTIER	193	Removes carriage returns to make long lines.
LOWCASE	193	Converts uppercase to lowercase throughout a file.
OLAC	194	Creates XML index files for the OLAC database
ORT	194	Converts Chinese characters.
PRAAT2CHAT	195	Converts PRAAT files to CHAT
QUOTES	195	Moves quoted material to its own tier.
REPEAT	195	Inserts postcodes to mark repeated utterances.
RETRACE	195	Inserts retrace markers.
SALTIN	195	Converts SALT files to CHAT format.
SILENCE	196	Converts utterances with LASTNAME to silence
TEXTIN	196	Converts straight text to CHAT format.
TIERORDER	196	Rearranges dependent tiers into a consistent order.
TRNFIX	196	Compares the %trn and %mor lines.
UNIQ	196	Sorts lexicon files and removes duplicates.

12.1 CHAT2CA

The CHAT2CA program will convert a CHAT file to a format that is closer to standard

CA (Conversation Analysis) format. This is a one-way conversion, since we cannot convert back to CHAT from CA. Therefore, this conversion should only be done when you have finished creating your file in CHAT or when you want to show you work in more standard CA format. The conversion changes some of the non-standard symbols to their standard equivalent. For example the #3_2 form of marking pauses is changed to the (3.2) form and speedup and slowdown are marked by inward and outward pointing arrows.

12.2 CHAT2ELAN

This program converts a CHAT file to the ELAN format for gestural analysis. For conversion in the opposite direction, use ELAN2CHAT. You can download the ELAN program from http://www.mpi.nl/tools/elan.html.

12.3 CHAT2XMAR

This program converts a CHAT file to the EXMARaLDA format for Partitur analysis. For conversion in the opposite direction, use XMAR2CHAT. You can download the EXMARaLDA program from http://www1.uni-hamburg.de/exmaralda/.

12.4 CHSTRING

This program changes one string to another string in an ASCII text file. CHSTRING is useful when you want to correct spelling, change subjects' names to preserve anonymity, update codes, or make other uniform changes to a transcript. This changing of strings can also be done on a single file using a text editor. However CHSTRING is much faster and allows you to make a whole series of uniform changes in a single pass over many files.

By default, CHSTRING is word-oriented, as opposed to string-oriented. This means that the program treats *the* as the single unique word *the,* rather than as the string of the letters "t", "h", and "e". If you want to search by strings, you need to add the +w option. If you do, then searching for *the* with CHSTRING will result in retrieving words such as *other, bathe,* and *there*. In string-oriented mode, adding spaces can help you to limit your search. Knowing this will help you to specify the changes that need to be made on words. Also, by default, CHSTRING works only on the text in the main line and not on the dependent tiers or the headers.

When working with CHSTRING, it is useful to remember the functions of the various metacharacters, as described in the metacharacters section. For example, the following search string allows you to add a plus mark for compounding between "teddy" and "bear" even when these are separated by a newline, since the underscore character matches any one character including space and newline. You need two versions here, since the first with only one space character works within the line and the second works when "teddy" is at the end of the line followed by first a carriage return and then a tab:

```
+s"teddy_bear" "teddy+bear” +s"teddy__bear" "teddy+bear"
```

Unique Options

+b Work only on material that is to the right of the colon which follows the tier ID.

+c Often, many changes need to be made in data. You can do this by using a text editor to create an ASCII text file containing a list of words to be changed and what they should be changed to. This file should conform to this format:

```
"oldstring" "newstring"
```

The default name for the file listing the changes is changes.cut. If you don't specify a file name at the +c option, the program searches for changes.cut. If you want to another file, the name of that file name should follow the +c. For example, if your file is called mywords.cut, then the option takes the form **+cmywords.cut**.

To test out the operation of CHSTRING with +c, try creating the following file called changes.cut:

```
"the" "wonderful"
"eat" "quark"
```

Then try running this file on the sample.cha file with the command:

```
chstring +c sample.cha
```

Check over the results to see if they are correct. If you need to include the double quotation symbol in your search string, use a pair of single quote marks around the search and replacement strings in your include file. Also, note that you can include Unicode symbols in your search string.

+d This option turns off a number of CHSTRING clean-up actions. It turns off deletion of blank lines, removal of blank spaces, removal of empty dependent tiers, replacement of spaces after headers with a tab, and wrapping of long lines. All it allows is the replacement of individual strings.

+l Work only on material that is to the left of the colon which follows the tier ID. For example, if you want to add an "x' to the %syn to make it %xsyn, you would use this command:

```
chstring +s"%mor:" "%xmor:" +t% +l *.cha
```

+q CHAT requires that a three letter speaker code, such as *MOT:, be followed by a tab. Often, this space is filled by three spaces instead. Although this is undetectable visually, the computer recognizes tabs and spaces as separate entities. The +q option brings the file into conformance with CHAT by replacing the spaces with a tab. It also reorganizes lines to wrap systematically at 80 characters.

+s Sometimes you need to change just one word, or string, in a file(s). These strings can be put directly on the command line following the **+s** option. For example, if you wanted to mark all usages of the word *gumma* in a file as child-based forms, the option would look like this:

```
+s"gumma" "gumma@c"
```

+w Do string-oriented search and replacement, instead of word-oriented search and replacement. CAUTION: Used incorrectly, the +w switch can lead to serious losses of important data. Consider what happens when changing all occurrences of “yes” to “yeah.” If you use this command with the +w switch included,

```
chstring +w +s"yes" "yeah" myfile.cha
```

every single occurrence of the sequence of letters y-e-s will be changed. This includes words, such as “yesterday,” “eyes,” and “polyester,” which would become “yeahterday,” “eyeah,” and “polyeahter,” respectively. However, if you omit the +w switch, you will not have these problems. Alternatively, you can surround the strings with spaces, as in this example:

```
chstring +w +s" yes " " yeah " myfile.cha
```

+x If you want to treat the asterisk (*), the underline (_), and the backslash (\) as the literal characters, instead of metacharacters, you must add this switch.

CHSTRING can also be used to remove the bullets in CHAT files that link to media. There are several steps in this procedure.

1. Open a file with bullets and expand bullets using escape-A.
2. Highlight one of the bullets, NOT the whole info inside the bullet.
3. Select the copy command to get an image of the bullet.
4. Move to the Commands window or open it with command-D.
5. In the Commands window type: chstring +s"
6. Then paste a bullet after the first quote. It will look like a blank space.
7. After this blank space, type an asterisk, then the bullet again, then another quote, then a space, and two more quotes and then the file name.
8. The resulting command should look like this with the bullet character indicated with the underline: `chstring +s"_*_" "" sample.cha`
9. Finally, just run the command on one file or a groups of files, as in *.cha.

In order to make this process easier, we have created a one-line file called bullets.cut which is located in the CLAN/lib/fixes folder. To remove all bullets, you can put this file into your working directory and type:

```
chstring  +crem_bullets.cut  t.cha
```

CHSTRING also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

12.5 COMPOUND

This program changes pairs of words to compounds, to guarantee more uniformity in morphological and lexical analysis. It requires that the user create a file of potential compound words in a format with each compound on a separate line, as in this example.

```
night+night
```

```
Chatty+baby
oh+boy
```

Whenever the program finds “night night” in the text, whether it be written as “night+night”, “night night” or “night-night,” it will be changed to “night+night”.

12.6 COMBTIER

COMBTIER corrects a problem that typically arises when transcribers create several %com lines. It combines two %com lines into one by removing the second header and moving the material after it into the tier for the first %com.

12.7 CP2UTF

CP2UTF converts code page ASCII files and UTF-16 into UTF-8 Unicode files. If there is an @Font tier in the file, the program uses this to guess the original encoding. If not, it may be necessary to add the +o switch to specify the original language, as in +opcct for Chinese traditional characters on the PC. If the file already has a @UTF8 header, the program will not run. The +c switch uses the unicode.cut file in the Library directory to effect translation of ASCII to Unicode for IPA symbols, depending on the nature of the ASCII IPA being used. For example, the +c3 switch produces a translation from IPAPhon. The +t@u switch forces the IPA translation to affect main line forms in the text@u format.

12.8 DATACLEAN

DATACLEAN is used to rearrange and modify old style header tiers and line identifiers.

1. If @Languages tier is found, it is moved to the position right after @Begin.
2. If @Participants tier is found, it is moved to the position right after @Languages.
3. If the tier name has a space character after ':', then it is replaced with tab. If the tier name doesn't have a following tab, then it is added. If there is any character after the tab following a speaker name, such as another tab or space, then it is removed.
4. Tabs in the middle of tiers are replaced with spaces.
5. If utterance delimiters, such as +..., are not separated from the previous word with a space, then a space is inserted.
6. if [...] is not preceded or followed by space, then space is added.
7. Replaces #long with ###.
8. The string "..." is replaced with "+...".

12.9 DATES

The DATES program takes two time values and computes the third. It can take the child’s age and the current date and compute the child’s date of birth. It can take the date of birth and the current date to compute the child’s age. Or it can take the child’s age and the date of birth to compute the current date. For example, if you type:

```
dates +a 2;3.1 +b 12-jan-1962
```

you should get the following output:

```
@Age of Child: 2;3.1
@Birth of Child: 12-JAN-1962
@Date: 13-APR-1964
```

Unique Options

+a	Following this switch, after an intervening space, you can provide the child's age in CHAT format.
+b	Following this switch, after an intervening space, you can provide the child's birth date in day-month-year format.
+d	Following this switch, after an intervening space, you can provide the current date or the date of the file you are analyzing in day-month-year format.

DATES uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

12.10 DELIM

DELIM inserts a period at the end of every main line if it does not currently have one.

12.11 ELAN2CHAT

This program converts ELAN files (http://www.mpi.nl/tools) to CHAT files. Use CHAT2ELAN for conversion in the opposite direction.

12.12 FIXBULLETS

This program is used to fix the format of the bullets that are used in CLAN to link a CHAT file to audio or video. Without any additional switches, it fixes old format bullets that contain the file name to new format bullets and inserts an @Media tier at the beginning of the file. The various switches can be used to fix other problems. The +l switch can be used to make the implicit declaration of second language source explicit. The +o switch can be used to change the overall start time for a file.

+b	merge multiple bullets per line into one bullet per tier
+oN	time offset value N (+o+800 means add 800, +o-800 means subtract)
+fS	send output to file (program will derive filename)
-f	send output to the screen or pipe
+l	add language tag to every word
+re	run program recursively on all sub-directories.
+tS	include tier code S

-tS	exclude tier code S
+/-2	+2 do not create different versions of output file names / -2 create them File names can be "*.cha" or a file of list of names "@:filename"

12.13 FIXIT

FIXIT is used to break up tiers with multiple utterances into standard format with one utterance per main line.

12.14 FIXMP3S

This program fixes errors in the time alignments of sound bullets for MP3 files in versions of CLAN before 2006.

12.15 FLO

The FLO program creates a simplified version of a main CHAT line. This simplified version strips out markers of retracing, overlaps, errors, and all forms of main line coding. The only unique option in FLO is +d, which replaces the main line, instead of just adding a %flo tier.

FLO also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the

12.16 INDENT

This program is used to realign the overlap marks in CA files. The files must be in a fixed width font such as FixedSysExcelsior.

12.17 INSERT

Programs such as STATFREQ, MLU, and MLT can use the information contained in the @ID header to control their operation. After creating the @Participants line, you can run INSERT to automatically create @ID headers. After this is done, you may need to insert additional information in these header tiers by hand.

12.18 LIPP2CHAT

In order to convert LIPP files to CHAT, you need to go through two steps. The first step is to run this command:

```
cp2utf –c8 *.lip
```

This will change the LIPP characters to CLAN’s UTF8 format. Next you run this command:

```
lipp2chat –len *.utf.cex
```

This will produce a set of *.utf.cha files which you can then rename to *.cha. The obligatory +l switch requires you to specify the language of the transcripts. For example "en" is for English and "fr" is for French.

12.19 LONGTIER

This program removes line wraps on continuation lines so that each main tier and each dependent tier is on one long line. It is useful what cleaning up files, since it eliminates having to think about string replacements across line breaks.

12.20 LOWCASE

This program is used to fix files that were no transcribed using CHAT capitalization conventions. Most commonly, it is used with the +c switch to only convert the initial word in the sentence to lowercase. To protect certain proper nouns in first position from the conversion, you can create a file of proper noun exclusions.

12.21 MAKEMOD

This program uses the CMU Pronouncing Dictionary to insert phonological forms in SAMPA notation. The dictionary is copyrighted by Carnegie Mellon University and can be retrieved from http://www.speech.cs.cmu.edu/cgi-bin/cmudict/ Use of this dictionary, for any research or commercial purpose, is completely unrestricted. For MAKEMOD, we have created a reformatting of the CMU Pronouncing Dictionary that uses SAMPA. In order to run MAKEMOD, you must first retrieve the large cmulex.cut file from the server and put it in your library directory. Then you just run the program by typing

```
makemod  +t* filename
```

By default, the program analyzes all speakers. However, you can control this using the +t switch. Also, by default, it only inserts the first of the alternative pronunciations from the CMU Dictionary. However, you can use the +a switch to force inclusion of all of them. The beginning of the cmulex.cut file gives the acknowledgements for construction of the file and the following set of

CMU	Word	full CMU	SAMPA
AA	odd	AAD	A
AE	at	AE T	{
AH	hut	HH AH T	@
AO	ought	AO T	O
AW	cow	K AW	aU
AY	hide	HH AY D	aI
B	be	B IY	b
CH	cheese	CH IY Z	tS
D	dee	D IY	d
DH	thee	DH IY	D
EH	Ed	EH D	E
ER	hurt	HH ER T	3

EY	ate	EY T	eI
F	fee	F IY	f
G	green	G R IY N	g
HH	he	HH IY	h
IH	it	IH T	I
IY	eat	IY T	i
JH	gee	JH IY	dZ
K	key	K IY	k
L	lee	L IY	l
M	me	M IY	m
N	knee	N IY	n
NG	ping	P IH NG	N
OW	oat	OW T	o
OY	toy	T OY	OI
P	pee	P IY	p
R	read	R IY D	r
S	sea	S IY	s
SH	she	SH IY	S
T	tea	T IY	t
TH	theta	TH EY T AH	T
UH	hood	HH UH D	U
UW	two	T UW	u
V	vee	V IY	v
Y	yield	Y IY L D	j
Z	zee	Z IY	z
ZH	seizure	S IY ZH ER	Z

The CMU Pronouncing Dictionary tone markings were converted to SAMPA in the following way. L0 stress was unmarked. A double quote preceding the syllable marked L1 stress and a percentage sign preceding the syllable being stressed marked L2 stress.

12.22 OLAC

This program goes through the various directories in CHILDES and TalkBank and creates an XML database that can be used by the OLAC (Online Language Archives Community) system to help researchers locate corpora relevant to their research interests.

12.23 ORT

ORT, if +c used, then this converts HKU style disambiguated pinyin with capital letters to CMU style lowercase pinyin on the main line. Without the +c switch, it is used to create a %ort line with Hanzi characters corresponding to the pinyin-style words found on main line. The choice of characters to be inserted is determined by entries in the lexicon files at the end of each word's line after the '%' character.

12.24 PRAAT2CHAT

This program converts files in the PRAAT format to files in CHAT format.

12.25 QUOTES

This program moves quoted material to its own separate tier.

12.26 REPEAT

REPEAT if two consecutive main tiers are identical then the postcode [+ rep] is inserted at the end of the second tier.

12.27 RETRACE

RETRACE inserts [/] after repeated words as in this example:

```
*FAT:   +^ the was the was xxx ice+cream .
%ret:   +^ <the was> [/] the was xxx ice+cream .
```

If +c is used then the main tier is replaced with an %ret tier and no additional %ret tier is not created.

12.28 RTFIN

This program is used to take data that was formatted in Word and convert it to CHAT.

12.29 SALTIN

This program takes SALT formatted files and converts them to the CHAT format. SALT is a transcript format developed by Jon Miller and Robin Chapman at the University of Wisconsin. By default, SALTIN sends its output to a file. Here is the most common use of this program:

```
saltin file.cut
```

It may be useful to note a few details of the ways in which SALTIN operates on SALT files:

1. When SALTIN encounters material in parentheses, it translates this material as an unspecified retracing type, using the [/?] code.
2. Multiple comments are placed on separate lines on a single comment tier.
3. SALT codes in square brackets are converted to CHAT comments in square brackets and left in the original place they occurred, unless the + symbol is added to the SALT code. If it is present, the code is treated as a postcode and moved to the end of the utterance when SALTIN runs. The CHSTRING program can be used to insert + in the desired codes or in all codes, if required.
3. Times in minutes and seconds are converted to times in hours:minutes:seconds.
4. A %def tier is created for coding definitions.

Unique Options

+h Some researchers have used angle brackets in SALT to enter comments. When the original contains text found between the < and the > characters this option instructs SALTIN to place it between [% and]. Otherwise, material in angle brackets would be coded as a text overlap.

+l Put all codes on a separate %cod line. If you do not select this option, codes will be placed on the main line in the [$text] format.

SALTIN also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

12.30 SILENCE

This program looks for utterances with the string Lastname and converts the linked segment of the related audio file to silence to preserve anonymity.

12.31 TEXTIN

The TEXTIN program is quite simple. It takes a set of sentences in paragraph form and converts them to a CHAT file. Blank lines are considered to be possible paragraph breaks and are noted with @Blank headers. To illustrate the operation of TEXTIN, here are the results of running TEXTIN on the previous three sentences:

```
@Begin
@Participants: T Text
*T:     the textin program is quite simple.
*T:     it takes a set of sentences in paragraph form and
        converts
        them to a chat file.
*T:     blank lines are considered to be possible paragraph
        breaks and
        are noted with @blank headers.
@End
```

There are no options that are unique to textin. However, it uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

12.32 TIERORDER

TIERORDER puts the dependent tiers into a consistent alphabetical order.

12.33 TRNFIX

This program compares the %trn line with the %mor line and notes discrepancies. This is useful in developing a training corpus for POSTTRAIN.

12.34 UNIQ

UNIQ is used to sort lexicon files into alphabetical order, while removing duplicates.

13 References

Aguado, G. (1988). Appraisal of the morpho-syntactic competence in a 2.5 month old child. *Infancia y Aprendizaje, 43,* 73-95.

Blake, J., Quartaro, G., & Onorati, S. (1970). Evaluating quantitative measures of grammatical complexity in spontaneous speech samples. *Journal of Child Language, 20,* 139-152.

Bohannon, N., & Stanowicz, L. (1988). The issue of negative evidence: Adult responses to children's language errors. *Developmental Psychology, 24,* 684-689.

Brainerd, B. (1982). The type–token relation in the works of S. Kierkegaard. In: R. W. Bailey (ed.) Computing in the humanities (pp. 97-109). Amsterdam: North-Holland.

Brown, R. (1973). *A first language: The early stages*. Cambridge, MA: Harvard.

Demetras, M., Post, K., & Snow, C. (1986). Feedback to first-language learners. *Journal of Child Language, 13,* 275-292.

Hausser, R. (1990). Principles of computational morphology. *Computational Linguistics*, 47.

Hickey, T. (1991). Mean length of utterance and the acquisition of Irish. *Journal of Child Language, 18,* 553-569.

Hirsh-Pasek, K., Trieman, R., & Schneiderman, M. (1984). Brown and Hanlon revisited: Mother sensitivity to grammatical form. *Journal of Child Language, 11,* 81-88.

Hoff-Ginsberg, E. (1985). Some contributions of mothers' speech to their children's syntactic growth. *Journal of Child Language, 12,* 367-385.

Klee, T., Schaffer, M., May, S., Membrino, S., & Mougey, K. (1989). A comparison of the age-MLU relation in normal and specifically language-impaired preschool children. *Journal of Speech and Hearing Research, 54,* 226-233.

Lee, L. (1974). *Developmental Sentence Analysis*. Evanston, IL: Northwestern University Press.

Malakoff, M.E., Mayes, L. C., Schottenfeld, R., & Howell, S. (1999) Language production in 24-month-old inner-city children of cocaine-and-other-drug-using mothers. *Journal of Applied Developmental Psychology, 20,* 159-180..

Malvern, D. D., & Richards, B. J., (1997). A new measure of lexical diversity. In: A. Ryan and A. Wray (Eds.) Evolving models of language. Clevedon: Multilingual Matters.

Malvern, D. D., & Richards, B. J. (in press). Validation of a new measure of lexical diversity. In B. v. d. Bogaerde & C. Rooijmans (Eds.), Proceedings of the 1997 Child Language Seminar, Garderen, Netherlands. Amsterdam: University of Amsterdam.

Moerk, E. (1983). *The mother of Eve as a first language teacher*. Norwood, NJ: Ablex.

Nelson, K. E., Denninger, M. S., Bonvilian, J. D., Kaplan, B. J., & Baker, N. D. (1984). Maternal input adjustments and non-adjustments as related to children's linguistic advances and to language acquisition theories. In A. D. Pellegrini & T. D. Yawkey (Eds.), *The development of oral and written language in social contexts.* Norwood, NJ: Ablex.

Nice, M. (1925). Length of sentences as a criterion of a child's progress in speech. *Journal of Educational Psychology, 16,* 370-379.

Pan, B. (1994). Basic measures of child language. In J. Sokolov & C. Snow (Eds.), *Hand-*

book of research in language acquisition using CHILDES (pp. 26-49). Hillsdale NJ: Lawerence Erlbaum Associates.

Richards, B. J., & Malvern, D. D, (1997a). Quantifying lexical diversity in the study of language development. Reading: University of Reading, The New Bulmershe Papers.

Richards, B. J., & Malvern, D. D. (1997b). type–token and Type-Type measures of vocabulary diversity and lexical style: an annotated bibliography. Reading: Faculty of Education and Community Studies, The University of Reading. (Also available on the World Wide Web at: http://www.rdg.ac.uk/~ehsrichb/home1.html)

Richards, B. J., & Malvern, D. D, (1998). A new research tool: mathematical modelling in the measurement of vocabulary diversity (Award reference no. R000221995). Final Report to the Economic and Social Research Council, Swindon, UK.

Rivero, M., Gràcia, M., & Fernández-Viader, P. (1998). Including non-verbal communicative acts in the mean length of turn analysis using CHILDES. In A. Aksu Koç, E. Taylan, A. Özsoy, & A. Küntay (Eds.), *Perspectives on language acquisition* (pp. 355-367). Istanbul: Bogaziçi University Press.

Rondal, J., Ghiotto, M., Bredart, S., & Bachelet, J. (1987). Age-relation, reliability and grammatical validity of measures of utterance length. *Journal of Child Language, 14,* 433-446.

Scarborough, H. S. (1990). Index of productive syntax. *Applied Psycholinguistics, 11,* 1-22.

Scarborough, H. S., Rescorla, L., Tager-Flusberg, H., Fowler, A., & Sudhalter, V. (1991). The relation of utterance length to grammatical complexity in normal and language-disordered groups. *Applied Psycholinguistics, 12,* 23-45.

Sichel, H. S. (1986). Word frequency distributions and type–token characteristics. *Mathematical Scientist, 11,* 45-72.

Snow, C. E. (1989). Imitativeness: a trait or a skill? In G. Speidel & K. Nelson (Eds.), *The many faces of imitation.* New York: Reidel.

Sokolov, J. L., & MacWhinney, B. (1990). The CHIP framework: Automatic coding and analysis of parent-child conversational interaction. *Behavior Research Methods, Instruments, and Computers, 22,* 151-161.

Templin, M. (1957). *Certain language skills in children*. Minneapolis, MN: University of Minnesota Press.

Wells, G. (1981). *Learning through interaction: The study of language development.* Cambridge, Cambridge University Press.