Tools for Analyzing Talk
Part 2: The CLAN Program

January 7, 2023

Brian MacWhinney
Carnegie Mellon University
https://doi.org/10.21415/T5G10R

When citing the use of TalkBank facilities, please use this reference to the last printed version of the CHILDES manual:

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

This allows us to systematically track usage of the programs and data through scholar.google.com.

1 Getting Started

This manual describes the use of the CLAN program, designed and written by Leonid Spektor at Carnegie Mellon University. The acronym CLAN stands for Computerized Language ANalysis. CLAN is designed specifically to analyze data transcribed in the CHAT format. This is the format used in the various segments of the TalkBank system. There are three parts to the overall TalkBank manual. Part 1 describes the CHAT transcription system. Part 2 (this current manual) describes the CLAN analysis programs. Part 3 describes the segments of the CLAN program that perform automatic morphosyntactic analysis.

1.1 Why you want to learn CLAN

If you are a researcher studying conversational interaction, language learning, or language disorders, you will want to learn to use CLAN, because it will help you address basic research questions and explore many different language types. If you are a clinician, CLAN can help you analyze data from individual clients and compare them against a large database of similar transcripts. For both these purposes, CLAN emphasizes the automatic computation of indices such as MLU, TTR, DSS, and IPSyn. It also provides powerful methods for speeding transcription, linking transcripts to media, sending data to automatic acoustic analysis, and automatic computation of a wide range of morphosyntactic features. For conversation analysts, CLAN provides the full range of Jeffersonian markings within a computationally clear framework. For all these purposes, CLAN is available free, as is the huge TalkBank database of transcripts compatible with CLAN analyses.

1.2 Learning CLAN

The first six chapters of this manual provide a basic introduction to CLAN. However, most users will find that it is best to begin learning about CLAN, CHILDES, and TalkBank by working through the screen tutorials to be found at https://talkbank.org/screencasts. Those screencasts will give you a clearer view of the actual process of data entry and analysis and then you can refer to the following six chapters for explicit descriptions of the methods.

1. Chapter 1 explains how to install and configure CLAN. This process has different steps, depending on whether you are using Windows or MacOS.
2. Chapter 2 explains how to access and use materials from the CHILDES and TalkBank homepages on the web.
3. Chapter 3 provides a tutorial on how to begin using CLAN commands.
4. Chapter 4 explains how to use the editor.
5. Chapter 5 explains how to link transcripts to media.
6. Chapter 6 provides advanced exercises for learning CLAN.

Ideally, you should work through all six chapters in that order. However, some users may wish to skip some sections. If you are not interested in transcribing new data, you can skip chapters 4 and 5 on the editor and linkage. People working with CA (Conversation Analysis) will probably not need to read chapter 3 on CLAN commands. The examples and analyses all focus on child language data. People working with other language types

such as aphasia, adult conversation, or second language may wish to use practice the exercises with CHAT files and media appropriate to those areas.

1.3 Installing CLAN – MacOS

Here is how to install and configure CLAN for MacOS:

1. Download the Mac version of CLAN from the link labeled "CLAN" at https://talkbank.org .
2. Click to open clan.dmg and then click to start the installer.
3. The system will complain about something from an unidentified developer and then you will have to open the Security and Privacy Panel in System Preferences, select General, open the lock, and click on "Anywhere".
4. CLAN will install in your Applications folder and your working directory will be: Applications/CLAN/work. For shared computers, there is also an option to install in ~/Applications.
5. Drag the CLAN file icon into the dock to create a link for easy access.
6. Go to System Preferences and select **Keyboard**. Check the two boxes there to use standard function keys and to show Keyboard Viewers in the menu bar.
7. Once CLAN is installed, you will probably also want to install the MOR grammar for your language. You can do this by selecting "Install MOR grammar" from the File menu, as described at https://talkbank.org/screencasts/mor-download.mp4. This will also set your path for MORLIB to the correct location.

1.4 Installation Problems – MacOS

If you are not able to start CLAN after installation or at some later point, the problem may be related to window positioning. Do you see on the left of menu bar next to Apple Icon the "CLANc" word? If you do, then CLANc did start. Then select "Window->Commands" menu or press command-D keys on keyboard. If you still do not see Commands window, then try menu "Edit->Reset options" and follow instructions in the pop-up dialog window. If it doesn't work, then quit CLANc, start it again and select "File->New" menu, next select "Edit->Reset options" menu again. If you do not see on the left of menu bar next to Apple Icon the "CLANc" word, then try deleting CLANc from Applications folder, empty trash and reboot your Mac. Next reinstall CLANc and try again.

1.5 Installing CLAN – Windows

Here is how to install and configure CLAN for Windows:

1. Download the Windows version of CLAN from the link labeled "CLAN" at https://talkbank.org .
2. CLAN will automatically install in c:/TalkBank and your working directory will be c:\TalkBank\CLAN\work. The installer will create shortcuts for CLAN and the \work folder.
3. Once CLAN is installed, you will probably also want to install the MOR grammar for your language. You can do this by selecting "Install MOR grammar" from the File menu, as described at https://talkbank.org/screencasts/mor-download.mp4 This will also set your path for MORLIB to the correct location.

2 Using the Web

In this chapter, we will first survey some of the community resources available at the TalkBank homepages. Then we will learn about how to download and play transcripts and linked media, and how to use the Browsable Database.

2.1 Community Resources

From the TalkBank homepage at https://talkbank.org, look at the community resource information at these links under System:

1. **Ground rules.** Whenever using TalkBank data, remember to cite the sources provided.
2. **Contributing New Data**. How to configure new research projects for eventual inclusion in TalkBank.
3. **IRB Principles**. We explain how to configure consent forms to specify the levels of confidentiality protection appropriate for your project.
4. **Programs**. Manuals and programs.

Then take a quick look at the homepages for AphasiaBank, BilingBank, CHILDES, SLABank, PhonBank, CABank, and other banks. Finally, look at the information about Google Group mailing lists and membership.

2.2 Downloading Materials

By default, CLAN materials will download to your desktop. You can then download additional materials, such as the manuals. Rather than printing out the long manuals, it is best to keep them in your /work folder and access them through Adobe Reader.

2.3 Using the Browsable Database

The Browsable Database facility allows you to playback transcripts with linked media directly from your browser. Here are the steps to follow:

1. Click on the Browsable Database link on the homepage for the TalkBank database you are using. When the new page opens, glance over the instructions. You can always come back to read these in detail later.
2. For an example in the CHILDES database, you can select Eng-UK/Forrester/biggirl.cha in the navigation area to the left. The screencase at https://talkbank.org/screencasts/browsable-watch.mp4 illustrates this further.
3. Study the display to get a sense of what a CHAT file looks like. There are headers for the first 11 lines and then the dialog begins on line 12. *E: is the child Ella and *F: is her father.
4. Each line is linked to the corresponding segment of the video, and both will play back over the web. Place your cursor on the right arrow on line 11 and either click or press "s". Usually, it takes a few seconds to establish the initial web connection, but playback is smooth after that.
5. If the video is stopped, pressing the "s" key starts it. If the video is playing, pressing the "s" key stops it. You can just follow along with continuous playback or you can

select certain segments to play.

2.4 Downloading Transcripts and Media

If you want to study transcripts more closely, you will probably want to download them, rather than playing through the Browsable Database. Using the Forrester transcripts as an example, here is how you do this:

1. At https://childes.talkbank.org click on **Index to Corpora**, then **Eng-UK** and **Forrester**.
2. Click on the link called **download transcripts** and the .zip file will download to your computer.
3. If it is not automatically unzipped, you should unzip it.
4. To download the media, click on **Link to Media Folder** and you can then download individual videos one by one. Downloading media takes a lot more time than downloading transcripts.

3 Tutorial

Once you have installed CLAN, you start it by double-clicking on its icon or its shortcut.

3.1 The Commands Window

After this, a window titled **Commands** opens and you can type commands into this window. If the window does not open automatically, then type *Control-d* (Windows) or ⌘*-d* (Macintosh). This window controls many of the functions of CLAN. It remains active until the program is terminated. The main components of the **Commands** window are the command box in the center and the several buttons. There is also some text in the bottom line giving you the data when your version of CLAN was compiled.

Sometimes the **Commands** window may disappear into the areas off the screen and then it can be difficult to drag it back. When this happens, you can go to the top menu **Edit** and pull down to **Reset Options** and the window should reappear.

3.1.1 Setting the Working Directory

The first thing you need to do when running CLAN is to set the **working** directory. The working directory is the place where the files you would like to work with are located. For the tutorial examples, we use the CLAN /examples directory as our **working** directory. To set the **working** directory, press the **working** button in the Command window and use the navigation facility to locate the **examples** directory inside the CLAN directory and press the **Select Current Directory** button.

After selecting your **working** directory, you will 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. You can also double-click on the actual name of the working directory to see and go back to other directories you have recently visited.

To test 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 in the **CLAN Output** window.

```
> 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
  ---- (more lines here)
  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.

3.1.2 Output and Mor Lib Directories

You do not need to worry about setting the LIB directory, because CLAN sets this to the /lib folder in the CLAN distribution by default. You typically also do not have to worry about setting your **output** directory, because it will be the same as your working directory. However, if you want output files to go to some other directory, you can use this button to navigate to some other output folder. If you want to undo that action, you can use the button to navigate back to your working directory and it will again become the default.

If you are using the Get MOR Grammar function in the CLAN File menu, it will download MOR to your desktop and set the path automatically. Alternatively, you may wish to control where to put your MOR grammars and then you need to set the **mor lib** button on your own for this.

3.1.3 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 in the **Commands** window. These commands can be automatically entered in 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.

3.1.4 The Progs Menu

The **Progs** menu gives you a list of CLAN commands you can run. Try clicking this button and then selecting the FREQ command. The name of the command will then be inserted into the **Commands** window.

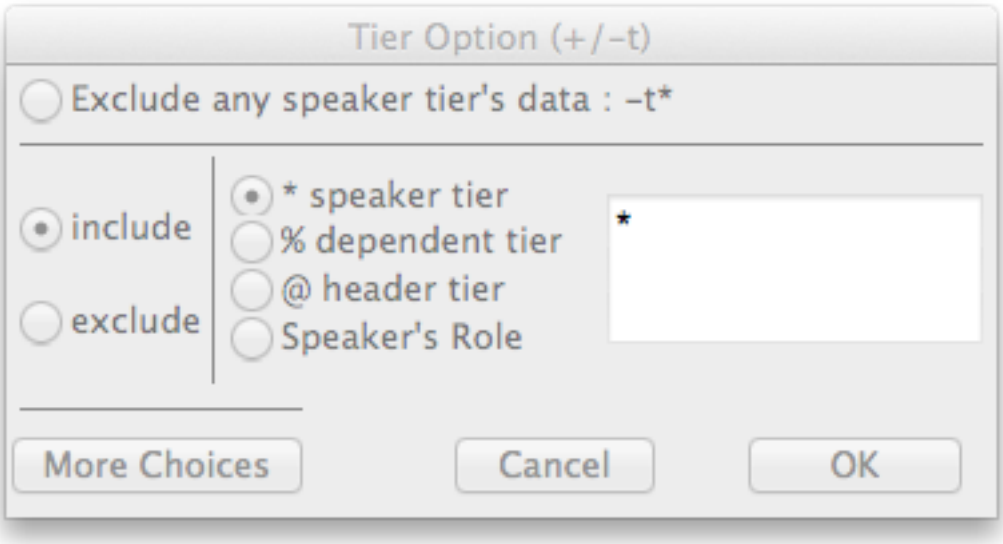
3.1.5 The FILE IN Button

Once you have selected the FREQ command, you now see that the **File 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. The files on the left are the items in your working directory. 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 radio button at the bottom right allows you to see only *.cha and *.cex files if you wish. 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. In this case, the @ represents the single file

"sample.cha".

3.1.6 The TIERS Button

This button will allow you to restrict your analysis to a certain participant. For this example, we will restrict our analysis to the child, who is coded as *CHI in the transcript, so we type "CHI" into the Tier Option dialog, leaving the button for "main tier" selected.



At this point, the command being constructed in the Commands window should look like this: freq @ +t*CHI If you hit the RUN button at the bottom right of the Commands window, or if you just hit a carriage return, the FREQ program will run and will display the frequencies of the six words the child is using in this sample transcript.

3.2 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, but you will find it useful when you are beginning. Alternatively, you can just type in commands directly to the **Commands** window. Let us 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 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 need 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.

3.2.1 The Asterisk Wildcard

CLAN uses several metacharacters or wildcards in commands. A very common wild card is the asterisk symbol (*) which is used to mark the presence of any string. 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
```

The asterisk wildcard 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.

3.2.2 Output Files

When you run a command with the +f option, the program will create an output file with the .cex extension. It drops the .cha extension from the input file and then adds a two-part extension to indicate which command has run 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.

3.2.3 Redirection

Instead of using the +f switch for output, you may sometimes want to use the redirect symbol (>). This symbol sends all of the outputs 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. There are three forms of redirection, as 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.

3.3 Sample Runs

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

3.3.1 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. To access the 0042.cha file, you need to change your working directory to the /transcripts folder inside the examples folder.

```
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. A segment of the output looks as follows:

```
----------------------------------------
*** File "0042.cha": line 2724. Keyword: bunny
*CHI:   0 .
*MOT:   see ?
*MOT:   is the bunny rabbit jumping ?
*MOT:   okay .
*MOT:   wanna [: want to] open the book ?
----------------------------------------
```

If you triple-click on the line with the three asterisks, the whole orginal transcript will

open with that line highlighted. Repetitions and retracing will be excluded by default unless you add the +r6 switch to the command.

3.3.2 Sample FREQ Run

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
```

In this file, the child uses the filler "uh" a lot, but that is ignored in the analysis. The output for this command is:

```
> freq  +t*CHI 0042.cha
freq +t*CHI 0042.cha
Sat Jun 14 14:38:12 2014
freq (13-Jun-2014) is conducting analyses on:
  ONLY speaker main tiers matching: *CHI;
****************************************
From file <0042.cha>
Speaker: *CHI:
  1 ah
  2 bow+wow
  1 vroom@o
------------------------------
    3  Total number of different item types used
    4  Total number of items (tokens)
0.750  Type/Token ratio
```

A statistical summary is provided at the end. In the above example, there were a total of 4 words or tokens used with only 3 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 3 divided by 4 or 0.750.

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 cludes all retraced material unless otherwise commanded. For example, given this utterance:

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

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

3.3.3 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 +t*MOT 0042.cha
mlu +t*MOT 0042.cha
Sat Jun 14 14:41:48 2014
mlu (13-Jun-2014) is conducting analyses on:
  ONLY dependent tiers matching: %MOR;
****************************************
From file <0042.cha>
MLU for Speaker: *MOT:
  MLU (xxx, yyy and www are EXCLUDED from the utterance and morpheme
counts):
   Number of: utterances = 511, morphemes = 1588
   Ratio of morphemes over utterances = 3.108
   Standard deviation = 2.214
```

Thus, we have the mother's MLU or ratio of morphemes over utterances (3.108) and her total number of utterances (511).

3.3.4 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 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
Sat Jun 14 14:44:21 2014
combo (13-Jun-2014) is conducting analyses on:
  ONLY speaker main tiers matching: *MOT;
****************************************
From file <0042.cha>
----------------------------------------
*** File "0042.cha": line 3034.
*MOT:   (1)kitty (1)kitty kitty .
----------------------------------------
*** File "0042.cha": line 3111.
*MOT:   and (1)kitty (1)kitty .
    Strings matched 2 times
```

3.3.5 Sample GEM and GEMFREQ Runs

GEM and GEMFREQ look at previously tagged selections or "gems" within larger

transcripts for further analyses. For example, we might want to divide the transcript by different social situations or activities. In the 0012.cha file, there are gem markers delineating the segment of the transcript that involves book reading, using the code word "book". By dividing the transcripts in this manner, separate analyses can be conducted on each situation type. Once this is done, you can use this command to compute a frequency analysis for material in these segments:

```
gemfreq +t*CHI +sbook 0012.cha
```

The output is as follows:

```
> gemfreq +sbook +t*CHI 0012.cha
gemfreq +sbook +t*CHI 0012.cha
Sat Jun 14 14:54:19 2014
gemfreq (13-Jun-2014) is conducting analyses on:
  ONLY speaker main tiers matching: *CHI;
  and ONLY header tiers matching: @Bg:; @Eg:;
****************************************
From file <0012.cha>
 24 tiers in gem "book":
      2 kitty
      2 no+no
      2 oh
      2 this
```

GEM and GEMFREQ are particularly useful in corpora such as the AphasiaBank transcripts. In these, each participant does a retell of the Cinderella story that is marked with @G: Cinderella. Using the three Kempler files, the following command will create three new files with only the Cinderella segment:

```
gem +sCinderella +n +d1 +t*PAR +t%mor +f *.cha
```

You can then run further programs such as MLU or FREQ on these shorter files.

3.4 Advanced Commands

This section provides a series of CLAN commands designed to illustrate a fuller range of options available in some of the most popular CLAN commands. With a few exceptions, the commands are designed to run on the Adler directory included in the /examples folder. So, you should begin by opening CLAN and setting your working directory to the Adler folder. Each command is followed by an English-language explanation of the meaning of each of the terms in the command, translating in order from left to right. You should test out each command and study its results. To save typing, you can cut cut each command from this document and paste it into the CLAN Commands window and then hit a carriage return.

Run KWAL on the Participant looking for "slipper" in all the files:

```
kwal +t*PAR +s"slipper*" *.cha
```

Run KWAL on the Participant looking for "because" in adler23a.cha:

```
kwal +t*PAR +s"because" adler23a.cha
```

Run KWAL on the Participant looking specifically for "because" transcribed as (be)cause (produced as "cause") in adler 23a.cha:

```
kwal +t*PAR +s"(be)cause" adler23a.cha +r2
```

Run KWAL on the Participant on the list of words in the whwords.cut file in the /examples/pos folder. You will need to copy that file into the /Adler folder.

```
kwal +t*PAR +s@whwords.cut adler23a.cha
```

Run KWAL on the Participant to exclude utterances coded with the post-code [+ exc] and create new files in legal CHAT format for all the files:

```
kwal -s"[+ exc]" +d  +t*PAR +t%mor +t@ +f *.cha
```

Run COMBO on the Participant to find all sequences of "fairy" followed immediately by "godmother" and combine the results from all the files into a single file:

```
combo +t*PAR +sfairy^godmother +u *.cha
```

Run COMBO on the Participant's %mor tier to find all combinations of infinitive and verb in adler01a.cha:

```
combo +s"inf|*^v|*" +t*PAR +t%mor adler01a.cha
```

Run MAXWD on the Participant to get the longest utterance in words in all files:

```
maxwd +g2 +t*PAR *.cha
```

Run EVAL on the Participant to get a spreadsheet with summary data (duration, MLU, TTR, % word errors, # utterance errors, % various parts of speech, # repetitions, and # revisions) in all the files. Add +o4 to get output in raw numbers instead of percentages.

```
eval +t*PAR +u *.cha
```

This program is similar to EVAL, but tailored for child data:

```
kideval +t*PAR +leng *.cha
```

Run MLU on the Participant, creating one spreadsheet for all files. Add -b to get mlu in words:

```
mlu +t*PAR +d +u *.cha
```

Run MLT on the Participant, creating one spreadsheet for all files. MLT counts utterances and words on a line that may include xxx (unlike MLU):

```
mlt +t*PAR +d *.cha
```

Run TIMEDUR on the Participant, creating a spreadsheet with ratio of words and utterances over time duration for all files:

```
timedur +t*PAR +d10 *.cha
```

Run GEM on the Participant, including the %mor line, using the "Sandwich" gem with lazy gem marking, outputting legal CHAT format for adler07.cha:

```
gem +t*PAR +t%mor +sSandwich +n +d1  adler07a.cha
```

Same thing, excluding irrelevant lines:

```
gem +t*PAR +t%mor +sSandwich +n +d1  -s"[+ exc]" adler07a.cha
```

Run GEM on the Participant main tier and %mor tier for the Sandwich "gem", using lazy gem marking, create a new file in legal CHAT format called "Sand" for all Adler files

```
gem +t*PAR +t%mor +sSandwich +n +d1 +fSand *.cha
```

Run VOCD on the Participant, output to spreadsheet only, and include repetitions and revisions in all the files:

```
vocd +t*PAR +d3 +r6 *.cha
```

Run CHIP to compare the Mother and the Child in terms of utterance overlaps with both the previous speaker (%chi and %adu, echoes) and their own previous utterances (%csr and %asr, self-repetitions) in chip.cha:

```
chip +bMOT +cCHI chip.cha  (the chip.cha file is in examples/progs)
```

Same thing, but excluding printing of the results for the self-repetitions:

```
chip +tMOT +cCHI -ns chip.cha
```

The next commands all use the FREQ program to illustrate various options.

Run FREQ on the Participant tier and get output in order of descending frequency for adler01a.cha:

```
freq +t*PAR +o adler01a.cha
```

Run FREQ on the Participant tier and send output to a spreadsheet for adler01a.cha. To open the spreadsheet, triple-click on stat.frq.xls:

```
freq +t*PAR +d2 adler01a.cha
```

Same, on all the files in Adler:

```
freq +t*PAR +d2 *.cha
```

Same, but only include Anomics:

```
freq +t@"ID=*|Anomic|*" +d2 *.cha
```

Run FREQ on the Participant tier and get type token ratio only in a spreadsheet for adler01a.cha:

```
freq +t*PAR +d3 adler01a.cha
```

Run FREQ on the Participant %mor tier and not the Participant speaker tier and get output in order of descending frequency for adler01a.cha:

```
freq +t%mor +t*PAR -t* +o adler01a.cha
```

Run FREQ on the Participant %mor tier for stems only (happily and happier = happy) and get output in order of descending frequency for adler01a.cha:

```
freq +t*PAR +t%mor -t* +s"mr-*,o-%" +o adler01a.cha
```

Learn how to use the +s switch for analysis of the %mor line

```
freq +sm
```

Learn how to use the +s switch for analysis of the %gra line

```
freq +sg
```

Run FREQ on the Participant tier, include fillers "uh" and "um", and get output in order of descending frequency for adler01a.cha:

```
freq +t*PAR +s+&uh +s+&um +o adler01a.cha
```

Run FREQ on the Participant tier and count instances of unintelligible jargon for adler01a.cha:

```
freq +t*PAR +s"xxx" adler01a.cha
```

Same, but adding +d to see the actual place of occurrence, then triple-click on any line that has a file name to open the original:

```
freq +t*PAR +s"xxx" +d adler01a.cha
```

Run FREQ on the Participant tier, counting instances of gestures for adler01a.cha:

```
freq +t*PAR +s&=ges* adler01a.cha
```

Run FREQ on the Participant tier, including repetitions and revisions, excluding neologisms (nonword:unknown target), and getting output in order of descending frequency for adler01a.cha. Add +d6 to include error production info. Add +d4 for type token info only.

```
freq +t*PAR +r6 -s"<*\* n:uk*>" +o adler01a.cha
```

Run FREQ on the Participant, searching for a list of words in a 0list.cut file you have created with multiple words searched per line, where multiple words do not have to be found in consecutive alignment, but must be in the same utterance, and merging output across all files:

```
freq +t*PAR +s@0list.cut +c3 +u *.cha
```

Same with +d added for outputting the original utterance:

```
freq +t*PAR +s@0list.cut +c3 +u +d *.cha
```

Here are some additional switches for making specific exclusions:

- add -s*** if you want to exclude words that were produced in error (coded with any of the [* errorcodes] on the main tier)
- add +r5 if you want to exclude any text replacements (horse [: dog], beds [: breads])
- add +r6 if you want to include repetitions and revisions

The final section in the description of the FREQ command gives many further detailed examples of how to use FREQ with the %mor and %gra tier.

3.5 Exercises

This section 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).They are included in the /examples/transcripts/ne20 folder. 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.

One approach to transcript analysis focuses on the computation of certain 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 /examples/transcripts/ne20 directory.

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.

3.5.1 MLU50 Analysis

The first CLAN analysis we will perform involves calculating MLU for each child on a sample of 50 utterances. By default, the MLU program runs on the %mor line that is already present in these files. This means that it computes the mean length of utterance in terms of morphemes, not words. Also 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 the /examples/transcripts/ne20 folder.

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 all the files to be analyzed ended with the same file extensions and were in the same directory; and 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. 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
Tue Jun 24 17:15:38 2014
mlu (24-Jun-2014) is conducting analyses on:
  ONLY dependent tiers matching: %MOR;
```

```
**************************************
From file <68.cha>
MLU for Speaker: *CHI:
  MLU (xxx, yyy and www are EXCLUDED from the utterance and morpheme
counts):
   Number of: utterances = 50, morphemes = 133
   Ratio of morphemes over utterances = 2.660
   Standard deviation = 1.595
```

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 1.595 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.

3.5.2 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 will run MAXWD on the five files in the ne20 folder and then MLU on the output of MAXWD. By default, MAXWD runs on the %mor line, rather than the main line.

```
maxwd +t*CHI +g1 +c5 +d1 *.cha
```

+gl Identify the longest utterances in terms of morphemes

+c5 Identify the five longest utterances

+d1 Output the data in CHAT format

***.cha** The child language transcripts to be analyzed

We then run MLU on the *.cex files that were the output of the previous command.

```
mlu *.cex
```

3.5.3 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 with only "0" on the main line. 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:

```
From file <68.cha>
MLT for Speaker: *MOT:
  MLT (xxx, yyy and www are EXCLUDED from the word counts, but are
INCLUDED in utterance counts):
    Number of: utterances = 356, turns = 227, words = 1360
   Ratio of words over turns = 5.991
   Ratio of utterances over turns = 1.568
   Ratio of words over utterances = 3.820
```

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 words/turn over mother words/turn. As the child begins to assume a more equal share of the conversational load, the MLT ratio should approach 1.00. For file 68.cha, this ratio is: 2.184 ÷ 5.991 = 0.365.

3.5.4 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 #. If you want to create a TTR based just on the full words on the main line, you will get this automatically when you run just *freq *.cha*. However, for TTR and other lexical diversity measures, we are interested not in whether the child uses plurals or past tenses, but how many different dictionary words (lemmas) she uses. Therefore, we wanted to count *cats* and *cat* as two tokens (i.e., instances) of the lemma *cat.* Similarly, we wanted to count *play* and *played* as two tokens of the lemma *play.* To make these distinctions correctly, we need to track the stem forms that are given on the %mor line. This means that we need to first use MOR and POST to create a %mor line for our transcript. The process of doing this is described in the MOR manual. For now, we will assume that the transcripts already have this %mor line. In that case, the command we use is:

```
freq +t*CHI +sm;*,o% +f *.cha
```

+t*CHI Analyze the child speaker only

+sm;*,o% Search for roots or lemmas and ignore the rest

+f Save output in a file

***.cha** Analyze all files ending with the extension .cha

The only new element in this command is +sm;*o%. The +s option tells FREQ to search for and count certain strings. The *m* tells the program to look at the %mor line. The semicolon says to look at the stem and then the *,o% says to ignore everything else. If you type *freq +sm* you will see explanations of the use of the +sm switch. The output generated from this analysis goes into five files. For the ne20/68.cha input file, the output is 68.frq.cex. At the end of this file, we find this summary analysis:

```
     81  Total number of different item types used
    242  Total number of items (tokens)
  0.335  Type/Token ratio
```

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

3.5.5 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.

Measure	Mean	SD	Range
MLU50	1.406	0.360	1.00-2.66
MLU5 longest	2.936	1.271	1.00-6.40
TTR	0.433	0.108	0.255-0.611
MLT Ratio	0.189	0.089	0.034-0.438

The distribution of MLU50 scores was quite skewed, with most children who produced at least 50 utterances falling in the MLU range of 1.00-1.30. As noted earlier, 17 of the 48 children failed to produce even 50 utterances. At this age most 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 MLU5 and for the MLT ratio were somewhat skewed toward the lower end, though not as severely as was MLU50.

Z-scores, or standard scores, are computed by subtracting the sample mean score from the child's score on a particular measure 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.

Child	MLU50	MLU5	TTR	MLT Ratio
14	0.26	0.21	1.65	-0.16
55	-0.30	-0.15	-0.36	-0.53
66	-0.16	-0.11	-0.64	-0.84
68	2.30	2.72	-0.86	1.98
98	-0.96	-0.74	-0.63	-0.08

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 certain 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). Overall, Subject 68 seems advanced on all measures except TTR. The subjects at the second and third quartile in terms of MLU (Subject 055 and Subject 066) 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, we have considered only three domains, as measured by four indices. 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 the number of words, or we might time-sample by examining the number of word types and word tokens the child produced in a certain 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 in the transcript prior to analysis; however, the CLAN analyses themselves would, for the most part, simply be variations on the techniques 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.

3.6 Further Exercises

The files needed for the following exercises are in the /examples/transcripts/ne20 and /examples/transcripts/ne32 folders. 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. Compare the MLT ratio for each activity and adult–child pair. Describe any differences you observe.

4 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, such as MS-Word. In fact, the basic functions of the CLAN editor and MS-Word are all the same. Some users say that they find the CLAN editor difficult to learn. However, on the basic level it is no harder than MS-Word. What makes the CLAN editor difficult is the fact that it is used to transcribe the difficult material of child language data with all its special forms, overlaps, and precise timings. These functions are outside of the scope of editors, such as MS-Word or Pages.

4.1 Screencasts

Use of the tutorial can be supplemented through the online screencasts for specific CLAN features found at https://talkbank.org/screencasts/ and on YouTube. These movies, created by Davida Fromm and Brian MacWhinney, show the use of specific CLAN functions in real time with real transcripts.

4.2 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 text 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 must 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 .cha extension, 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 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 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 as in Word or Pages. Functions like scrolling, highlighting, cutting, and pasting also work in the standard way. You should try out these functions right away. Use keys and the scroll bar to move around in the sample.cha file. Cut and paste sections and type a few sentences, just to convince yourself that you are already familiar with the basic editor functions.

4.3 File, Edit, Format, and Font Menus

The functions of opening file, printing, cutting, undoing, and font changing are the same as in Pages or Word. 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. This function works on Mac and older Windows systems. However, this function will not work on versions of Windows after about 2016, because it relies on QuickTime, which is no longer supported. In the newer version of CLAN for OSX, there is no longer a separate Font menu and fonts are set through an option in the Format menu.

4.4 Mode Menu

The top section of this menu provides access to three important editor modes described elsewhere in this manual: entering Coder Mode, setting CHAT mode, and showing line numbers. The bottom section of the menu allows you to start up these editor functions, which are also described elsewhere in this manual: continuous playback, play bullet media, transcribe sound or movie, check open file, disambiguate tier, expand bullets, and send to sound analyzer.

4.5 Default Window Positioning, Size, and Font Control

When CLAN starts up it will open a new Commands window and a new Text window in the same position it used when you last ran CLAN. If you want to change the position or size of a window, you can move it and resize it. The information regarding the size and position of a given window is stored in a non-visible @Window line at the beginning of the file. If you move and/or resize the window and save it, that information will be updated so that the next time you open the file, it will be in the new size and position.

If you open a new text window, you can position and size it and close it. When you then open another new window again, it will go to that position and size. However, this will not impact already existing files, because they use the information in their @Window line. Repositioning also works in the same way for the Commands window, but you cannot

resize the Commands window.

If you accidentally align the bottom of a window with the bottom of your screen, the last lines of the window will be cut off. This can make it difficult to use functions like esc-2 for disambiguation or Coder's Editor. To solve this problem, just open a new window in a higher position, close your earlier window, and then reopen.

You may also find it useful to have bottom and side scroll bars always visible. On MacOS, you can set this option to "always visible" in System Preferences/General.

When starting up video playback, it can be the case that the movie window occupies too much of the screen. To size it properly, you can click on the green button in the top of the QuickTime video window and the window will be resized to the smallest dimension. Then you drag on the botton right corner to expand it to the size you wish.

The system for controlling the default Font depends on your operating system. On Windows (PC) there is a Font menu under View. You use the **Set Font** option to set the text window font, and the **Set Commands Font** to set the Commands window font. The option to **Set Default Font** is only needed in rare cases when no default font had yet been selected. When using the **View** pulldown to change font or size, both the font and the size must be selected. If you select only one, no change will be made. On Macintosh, you can use the **Size/Style** menu to control the font. To save your font selection, you must not only save and quit the file, but also quit and then restart CLAN, because information about font size is stored not in the file, but in CLAN preferences.

4.6 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 @Options line, as in this example:

```
@Options:     CA
```

By default, CA files will use the CAfont, because the characters in this font have a fixed width, allowing the INDENT program to make sure that CA overlap markers are clearly aligned. When doing CA transcription, you can also select underlining and italics, although bold is not allowed, because it is too difficult to recognize. Special CA characters can be inserted by typing the F1 function key followed by some letter or number, as indicated in a list that you can find by selecting **Special Characters** under CLAN's **Windows** menu. The full list is at https://ca.talkbank.org/codes.html .

The F1 and F2 keys are also used to facilitate the entry of special characters for Hebrew, Arabic, and other systems. These uses are also listed in the Special Characters window. The raised h diacritic is bound to F1-shift-h and the subscript dot is bound to F1-comma.

4.7 Setting Special Colors

In CLAN for Windows, but not for macOS, you can set the color of certain tiers to improve the readability of your files. To do this, select the **Color Keywords** option in the **Size/Style** pulldown menu. 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.

4.8 Searching

In the middle of the **Edit** pulldown 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 that same search multiple times. The **Go To Line** command allows you to move to a particular line number. The **Replace** command brings up a dialog like the Find dialog. However, this dialog allows you to find a certain string and replace it with another one. You can replace some strings and not others by skipping over the ones you do not want to replace with the **Find-Next** function. When you need to perform a large series of different replacements, you can set up a file of replacement forms and use it by pressing the **from file** button. You then are led through the words in this replacement file one by one. The form of that file is like this:

```
"String_A"      "Replacement_A"
"String_B"      "Replacement_B"
```

4.9 Send to Sound Analyzer

This option under the **Mode** menu allows you to send a bulleted sound segment to Praat or Pitchworks. You choose which analyzer you want to use by an option under the **Edit** menu. The default analyzer is Praat. The bullets must be formatted in the current format (post 2006). If you have a file using the old format, you can use the FIXBULLETS program to fix them. If you are using Praat, you must first start up the Praat window (download Praat from http://fon.hum.uva.nl/praat) and place your cursor in front of a bullet for a sound segment. You must first play the selected bullet at least once to open the media window. Then, selecting **Send to Sound Analyzer** sends the bullet information of the clip to the Praat window for further analysis. After the media window is open, you just need to place your text cursor after the bullet when sending another clip's bullet information to Praat. 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
```

4.10 Tiers Menu Items

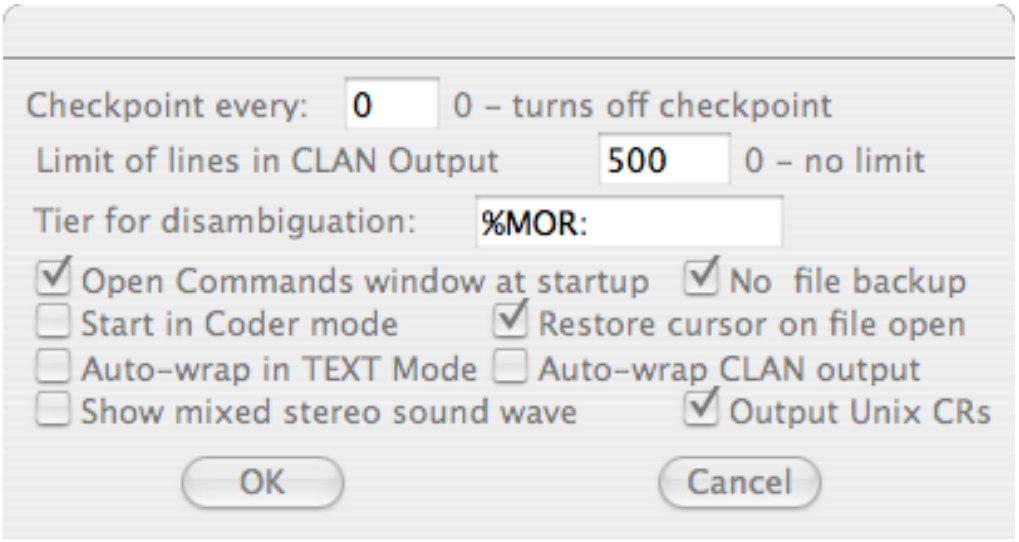
When you open a CHAT file with an @Participants line, the editor looks at each of the participants in that line and inserts their codes into the **Tiers** menu. You can then enter the name quickly, using the commands listed in that menu. If you make changes to the @Participants line, you can press the **Update** button at the bottom of the menu to reload new speaker names. As an alternative to manual typing of information on the @ID lines, you can enter information for each participant separately using the dialog system that you start up using the **ID Headers** option in the **Tiers** menu.

4.11 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 your **Lib** directory to the place where the depfile.cut file is located. On Windows, this should be c:\TalkBank\CLAN\lib. CHECK can also be run from the command line using a command such as: check *.cha. See the section on CHECK in the command descriptions of this manual for more details. The command line version of CHECK can spot a few additional problems that cannot be detected by the version that operates inside the editor.

4.12 Preferences and Options

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



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 also 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.

4.13 Coder Mode

Coder Mode is an advanced editor feature that 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] barry.cha 1
```

The [E] entry indicates that you are in editor mode and the [chat] entry indicates that you are in CHAT Mode. To begin coding, you first want to set your cursor on the first utterance you want to code. 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. You can use the barry.cha file in the /examples/transcripts folder. 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 (always release the escape key before entering the next key). You will be asked to load a codes file. Just navigate to the /examples/coder/ directory and select one of the demo codes files beginning with the word "code." We will use codes1.cut for our example.

4.13.1 Entering Codes

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 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 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. On Macintosh, use the command key instead of the control key. Remember to release the esc key before the next character.

```
Command Function
esc-c         finish current code
esc-c (again) finish current tier
control-z        undo
control-t or F1  finish current tier and go to next
esc-t         restrict coding to a particular speaker
esc-esc       go to the next speaker
esc-s         show subcodes under cursor
```

4.13.2 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. To do this, open up a new file using command-N. When editing this new 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 refer to the sample codes files in the /examples/coder folder. 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
```

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 the codes at a particular level

have the same codes symmetrically nested within them. For example, consider the codes-basic.cut file:

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

The spaces in this file must be spaces and not tabs. The line with $MOT begins with a space. Then there is the quote sign, followed by one more space. There are two spaces before :POS, because that code appears in the second field. There are three spaces before :Que, because that code appears in the third field. There must be a tab following the colon on the %spa: tier, because that code needs to be inserted in the actual output in the CHAT file. 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 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 /coder/codeserr.cut file.

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.

5 Media 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. The second field in the @Media header tells whether the media is audio, video, or missing. Each transcript should be associated with one and only one media file. For media linkage to work, it is crucial to include the @Media line before you begin linking, and the media file name must exactly match the transcript file name.

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, 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 file has been linked to audio or video, it is easy to playback the interaction from the transcript using “Continuous Playback” mode (esc-8, remember to always release the escape key before typing the next key). 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 exporting to partitur editors. This chapter describes each of these six methods and leaves it up to the individual researcher which of these methods is best for his or her project.

5.1 Media Formats

To use any of these methods, you need to have a digitized audio or video file. Audio files can be in either WAV (.wav) or MP3 (.mp3) format. For TalkBank, we recommend the CD quality WAV format which is 16-bit, stereo, with sampling at 44.1 Khz. This rate should be adquate for human speech. When compressing WAV files to MP3 for TalkBank

in Amadeus Pro, you should use CBR (Constant Bit Rate), Best quality, 320 kbps bitrate, Id3v2.4 tags, automatic sample rate, and no joint stereo.

For use in TalkBank browser web display and editing with CLAN, video files should be in MP4 (.mp4) format. Resolution can be 720x480 or 960x640. Very high resolution is not good for web delivery and lower resolution can be too grainy.

5.2 Sonic Mode

Sonic Mode involves transcribing from a sound waveform. This mode can also be used with a video file. In that case, before beginning, Sonic Mode extracts the audio track from the video file. If you do not specifically select the Sonic Mode option, you can still edit video directly without reference to the audio, as described in the section on Video Linking below.

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. If it stays highlighted, you can replay it by holding down the command key and clicking the mouse. On Windows, you use the control key, instead of the command key. At this point, it does not matter where in the bottom waveform window your cursor is positioned. But, if you click on the bullet in the text window, then you will play the sound linked to the bullet on which you click in that window.
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. As a side effect, once you expand or contract an area, the sound will also play.
3. **Transcription.** While you are working with the waveform, you can repeatedly play the sound by using command-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. **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.
5. **Adjusting start and stop times**. You can use four keyboard shortcuts to adjust the start and stop time of a segment in 25 msec intervals. To move the start time earlier, use control-leftarrow. To move the start time later, use control-rightarrow. To move the end time earlier, use shift-leftarrow. To move the end time later, use shift-rightarrow.
6. **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 command-I (insert time code). If you want to change the value of a bullet already in the transcript, you do the same thing while your cursor is inside or right next to the bullet in the transcript window.

7. **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. For highly accurate border placement, use a very wide horizontal display. The **+V** and **-V** buttons allow you to control the amplitude of the waveform.
8. **Scrolling.** At the bottom of the sound window is a scrollbar 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).
9. **Utterance -> Waveform Display**. To highlight the section of the waveform associated with an 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 command-click.
10. **Waveform -> Utterance Display**. Correspondingly, you can double-click an area of the waveform and, if there is a corresponding bullet in the transcript, then the line with that bullet will be highlighted.
11. **Undo**. If you make a mistake in linking or selecting an area, you can use the Undo function with command-Z to undo that mistake.
12. **Time duration information**. Just above the waveform, you will see the editor mode line. This is the black line that begins with the date. If you click on this line, you will see three additional numbers. The first is the beginning and end time of the current bullets-1 and F5bullets-2window in seconds. The second is the duration of the selected part of the waveform in hours:minutes:seconds.milliseconds. The third is the position of the cursor in seconds.milliseconds. If you click once again on the mode line, you will see sampling rate information for the audio file.

5.3 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.

5.3.1 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 in the same directory.
2. Set your *working* directory as the location of the video and transcript.
3. Open the CLAN transcript.
4. Type esc-L to make sure that your transcript passes CHECK. If not, fix any problems.
5. Insert an @Media header after the last @ID line, as described at the beginning of this chapter.

6. Place your cursor somewhere within the first utterance.
7. Click on Mode, Transcribe Sound or Movie or just type F5.
8. When CLAN asks you for the media, click on the audio or video file you want to transcribe.
9. The movie or audio will automatically start playing in Quicktime (Mac) or Windows Media Player (PC). 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.
10. If you get off at any point in time, click on the video window and the video will stop running.
11. 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 your cursor is. As you press the spacebar, the new bullets will overwrite the old ones.
12. After you have spent some time inserting bullets, click file, save. The bullets will be saved into your transcript.
13. 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.

5.3.2 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. Make sure that your media file is in your working directory along with your new blank file.
3. Enter the @Begin, @Languages, and @Media headers.
4. Go to "Mode," and select "Transcribe Sound or Movie F5."
5. Find the sound or movie clip you want to transcribe.
6. When you click on it, the sound or movie will open in another window.
7. Immediately, start pressing the spacebar at the end of each utterance. This will insert a bullet into the blank transcript.
8. When you are finished inserting the bullets, save the file.
9. Then, one at a time, Crtl-click on the bullets and transcribe them.
10. 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 righthand corner. You can also expand the bullets using esc-A and type it in manually.

11. Save frequently.
12. 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."

5.3.3 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. We can refer to this as sparse annotation of "naked" video. 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 second bullet length press OK button.
5. Start F5 mode by pressing the F5 key.
6. CLAN will start playing.
7. When you hear what you want to comment on 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.

5.4 Video Linking

To achieve maximum compatibility with CLAN and web-based video playback, we recommend using the H.264 format, which is now the default for high-definition. This is the format used by all new recording devices. Older video can be converted to H.264 format using programs like iSKySoft Video Converter or Handbrake. Additional information about video recording can be found at http://www.talkbank.org/info/dv.html.

When starting up video playback, it can be the case that the movie window occupies too much of the screen. In order to size it properly, you can click on the green button in the top of the QuickTime video window and the window will be resized to the smallest dimension. Then you drag on the botton right corner to expand it to the size you wish.

To learn how to do video linking, you should refer to the screencasts at http://talkbank.org/screencasts. 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 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

5.5 Walker Controller

The Walker Controller facility is based on a design from 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:

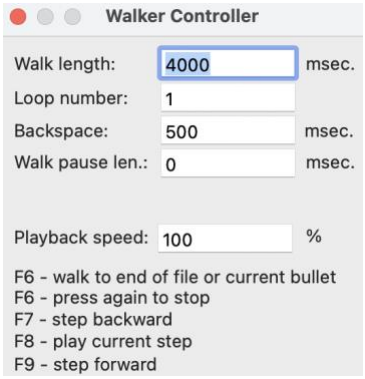
1. walk length: This sets how long a segment you want to have repeated.
2. loop number: If you set 3 loops, for example, the program plays each step three times before moving on.
3. backspace: The amount of rewinding in milliseconds at the end of each loop.

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 these options in the "Walker Controller" dialog that you open under the Window menu. To establish the correct link between your transcript and the media, you need to place the @Media line into your transcript, as described at the beginning of this chapter. Once you open the Walker Controller and thereby enable SoundWalker, the functions described above become enabled.



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.

5.6 Playback Control

Once a transcript has been linked, you will want to study it through playback. Basic playback uses the esc-8 command for continuous playback and command-click for playback of a single utterance. You can also use esc-9 to playback only the segments marked with time codes and ignore the silence in between.

5.7 Multiple Video Playback

If you have recorded an interaction using several cameras, and if the videos you have created are all synchronized in time, you can switch playback back and forth from these

multiple video files using a single CHAT transcript. To do this, you should name your videos with a constant first part of the name and then add an additional variable part to distinguish each video. For example, you could have "scale01" as your constant name and then the three videos would have the names scale01-1.mov, scale01-2.mov, and scale01-3.mov. If this is your naming convention for the three videos, then you would have these two lines in your *.cha transcript file:

```
@Media: scale01, video
@Videos:      1, 2, 3
```

If you want somewhat more mnemonic names, you could name the files as scale05-1center, scale05-2left, and scale05-3right and then the @Videos line would be:

```
@Videos:      1center, 2left, 3 right
```

When you open the transcript and start playback using continuous playback with esc-8 or some other method, the default video will be the first one in the list. To switch to playback from another video clip, stop the playback and type F3 followed by the number of the video you want to play. For example, if you want to shift playback from scale05-1 to scale05-3, you should type F3-3. A sample file with three videos for testing out this process can be downloaded from http://talkbank.org/resources/F3.

5.8 Manual Editing

You can use manual editing of the bullets to modify playback in three ways. The first way, which is very often quite important and practical, is to simply open the bullets using esc-A and modify the numbers using your own impressions of the length of the sound. You can modify either number and then check your work by playing the new alignment using command-click.

The second method uses the four key combinations described in the point on adjusting time values in the section on the Sonic Mode. To make this work, place your cursor right next to a bullet, and then these same four functions will also work even if the waveform window is not open. If you open a bullet and hit one of these key combinations, you will hear the audio and see the time values changing inside the bullet. Currently (August 2021) this method is not yet implemented for CLANc.

5.9 Video Skipping

The third method for manual editing applies in fewer cases. It is useful 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 need to listen through this untranscribed material during continuous playback. To implement forced skipping, you should open your bullets using esc-A. First, you need to trim the end of the utterance before the region you want to skip, so that it ends right before the skipped part. Then go to the end of the bullet after the skip and adjust the beginning of that bullet to begin just after the region to be skipped. Finally, insert a dash after the second time value in that bullet. For example, you might change 4035_5230 to 4035_5230-. Then close the bullets and save the file. You can test the marking by going to the utterance before the skip and playing through with esc-8. The playback should skip from the first utterance to the next one without playing the intervening skipped material.

5.10 Audio Anonymization

To deidentify the segments of the audio track in a recording, you can use Audacity or Amadeus Pro. In Amadeus Pro, this function is called *Generate Silence* and it is found under the *Effects* menu. You highlight the area to be silenced and then select that option.

In this case, you would still want to replace the last name with the *Lastname* in the transcript. Moreover, if you have been linking the transcript to the media, you can use the time values in milliseconds in the linkage bullets to locate the stretch to be silenced in the media. To find the right place in Amadeus, you will want to change the time display at the bottom of the waveform window to milliseconds. You do this by going into *Preferences* and selecting *Units* and entering this string in the *Format* box: |1/1000:3|

If you need to silence audio within a video recording, you can download the Subler app for Mac from https://bitbucket.org/galad87/subler/downloads/. You should make a copy of your movie to use later. Then you drag your movie onto the Subler icon, delete the video track, and save. Then drop this version onto Amadeus Pro, silence the stretches of the audio that you need to deidentify, and save. Then you open the original video in Subler, delete its audio and drag the output from Amadeus into the Subler window. You can find a screencast illustrating this at https://talkbank.org/share/irb/subler.mov. An advantage of this method is that saving the result is quite quick.

Alternatively, you can silence directly in iMovie for Mac. Instructions for this can be found on YouTube at https://www.youtube.com/watch?v=ShrDFjdvB1w. You can hold down the "R" key as an easier way of selecting an area to silence. An advantage of this method is that you can watch the video as you silence the audio. A disadvantage is that saving large video files takes several minutes.

6 Other Features

6.1 Supplementary Commands

The chapter on Supplementary Commands lists several basic commands for working with files. They are ***batch*** for creating batch files, ***cd*** for changing directory, ***dir*** for listing the files in a directory, ***info*** for getting a list of possible commands, ***ren*** for renaming files, and ***rm*** for removing or deleting files. See the descriptions in that chapter for details.

6.2 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. To see a list of all the possible commands in CLAN, type ***info***. 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.

6.3 Help for the +sm and +sg switches

There is a complex system for creating searches of the %mor and %gra lines. You can best understand the patterns used in this system by typing:

```
freq +sm
```

The patterns for the +sg switch are much less complicated, but you can see these by typing:

```
freq +sg
```

These patterns for +sm and +sg can also be used with KWAL and COMBO.

6.4 Keystroke Listing

It is possible to get a complete list of keystroke bindings for CLAN commands by typing esc-h. This creates a file called keys_list.cut that you can then read, save, or print out.

6.5 Macros

CLAN can store up to 10 Macros. Typing esc-n will open a dialog that lets you assign strings to numbers. For example, you might want to assign the string %spa: as Macro #1. You would type %spa: into the box next to Macro String: and then enter the number "1" above. Then you could insert this string by typing control-w-1. You will also see that just typing control-w pops up a list of all the macros you have assigned.

6.6 Aliases

If you make frequent use of a CLAN command with specific switches, you can save time and memory and increase reliability by creating an alias. For example, this alias

```
trim    kwal +t* +t@ +t% +d +f
```

will save you from having to remember all the details of how to configure KWAL to remove a given dependent tier. So, if you then type:

```
trim –t%mor *.cha
```

This will operate as if the command were really:

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

Note that it saves you from having to type the first 7 arguments, but you still have to type the last *.cha

If you want to remove both %mor and %gra lines, you can then just type:

```
trim –t%mor –t%gra *.cha
```

This trim alias is already included in the 0aliases.cut file in CLAN/lib/fixes. If you type the word "trim" by itself, CLAN will give you the usage message. The other standard alias is

```
chat2text flo +cr +t*
```

We will add additional aliases from time to time. However, if you want to create you own aliases, then you should create a file called aliases.cut that you put into the CLAN/lib folder. The format of that file is to have one command per line with the alias at the beginning of the line without spaces, then a space, and then the full CLAN command.

6.7 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 designed to do. 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 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. 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.

6.8 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 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.

6.9 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.

6.10 Types of CLAN Commands

There are seven types of CLAN commands:

1. **Analysis commands** are the basic commands for searching and corpus analysis.
2. **Profiling commands** put a large number of analysis and profiling commands into a single command package, often comparing a file against a database standard.
3. **Format conversion commands** convert from other formats to CHAT and from CHAT to other formats.
4. **Reformatting commands** are used to add features to transcripts which have passed CHECK and are in good CHAT format.
5. **Format repair commands** are used to rework files into CHAT format.
6. **Supplementary commands** are for file operations such as renaming or deleting files.
7. **Morphosyntactic analysis commands** serve to create a %mor tier for part-of-speech tagging and a %gra tier for grammatical relations tagging.

The first six of these command types are described in detail in the next six chapters. The morphosyntactic commands are described in the separate MOR manual which is downloadable from the talkbank.org website.

7 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. To go directly to any command in this click, click on the page number.

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 in the MOR manual. In addition, CLAN includes a large group of Utility commands that are described in the chapter on Utility Commands.

The best way to see a complete list of options for a command is to type the names of the command followed by a carriage return in the Commands window. For example, if you type just the word chip, you will see a list of all the available options for CHIP. You can see a list of all available commands by typing "info".

RELY	105	Measures reliability across two transcriptions.
SCRIPT	108	Compares transcripts to target scripts
TIMEDUR	108	Computes time durations of utterances and pauses
VOCD	110	Computes the VOCD lexical diversity measure.
WDLEN	115	Computes the length of utterances in words.

7.1 CHAINS

CHAINS is used to track sequences of interactional codes. These codes must be entered by hand on a single specified coding tier. 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 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.

7.1.1 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
```

7.1.2 Unique Options

CHAINS has the following unique 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 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.

7.2 Chatter

The Chatter program is not included in CLAN. However, it is needed for full validation of CHAT transcripts. When creating a new CHAT transcript, it is easiest to use CHECK on individual files for validation, as described in the next section. Often users submit corpora that have only been validated by CHECK, and then the TalkBank staff will run Chatter for final validation. However, it is best if users can run Chatter themselves. Chatter can be downloaded from https://talkbank.org/software/chatter.html. That page also provides additional instructions for running Chatter. Chatter only runs on complete folders, not on individual files, although it analyzes each file in the specified folder or folder hierarchy.

After running, Chatter creates a file called 0errors.cut in each folder. It is possible to double-click on errors reported in that file to open the site of the error in the original file for correction. If necessary, you can use the column number given for the error to find its precise location. After correcting errors, you can run Chatter again and there will hopefully be fewer or no errors. Once all errors detected by Chatter are corrected, a corpus is ready for inclusion in one of the TalkBank databases.

7.3 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.

7.3.1 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 inside the /lib folder 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.

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.

7.3.2 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.

7.3.3 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 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.

7.3.4 Restrictions on Word Forms

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.

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. 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. You will also want to 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.

7.3.5 Unique Options

CHECK has the following 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 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.

7.4 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.

7.4.1 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's that?
*CHI: hat.
*MOT: a hat!
*CHI: a hat.
*MOT: and what's this?
*CHI: a hat !
*MOT: yes that's 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's 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's 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's the hat .
```

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

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, by using 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 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.

7.4.2 The CHIP 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
$SUBST	pairs with substitutions, but no additions or deletions
$EXPAN	pairs with additions, but no deletions or substitutions
$EXP_SUB	pairs with additions and substitutions, but no deletions
$REDUC	pairs with deletions, but no additions or substitutions
$RED_SUB	pairs with deletions and substitutions, but no additions
$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's the hat .
%asr:   $NO_REP $REP = 0.00
%adu:   $EXA:hat $ADD:yes-that's-the $DEL:that-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's-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.

7.4.3 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 in /examples/pos/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's the hat.
%adu:   $EXA:that $EXA:hat $ADD:yes $SUB:the $MADD:'s $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's* 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.

7.4.4 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 exactly matching 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.

IMITAT Index of total imitativeness, using 4 indices above. It is this sum:

EXACT+EXPAN+REDUC+SUBST

7.4.5 Unique Options

CHIP has the following 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 which speaker is taken to be the source. If you want to study the "child" as respondent, you would focus on the %chi line. If you want to see focus on the "adult" as respondent, you would focus on the %adu line.

+c Specify that speaker ID S is a "child." The speaker does not actually have to be a child. The "c" simply indicates which speaker is taken to be the "response". If you want to study the "child" as respondent, you should focus on the %chi line. If you want to see focus on the "adult" as respondent, you should focus on the %adu line.

+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 or metacharacters 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 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.

7.5 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.

7.5.1 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:

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 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 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."

7.5.2 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 ! 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 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*.

7.5.3 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 followed by articles by using this switch:

```
+s@preps.cut^@arts.cut
```

To use this form, you first need to create a file of prepositions called "preps.cut" with one preposition on each line and a file of articles called "arts.cut" 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 another 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
```

7.5.4 COMBO for %mor and %gra sequences

You can use the +d7 switch in COMBO, FREQ, and KWAL to search for matches between the %gra and %mor lines, because these two lines are in one-to-one correspondence. The following command would search for cases in which a SUBJ on the %gra line is followed by a verb on the %mor line:

```
combo +d7 +sg|SUBJ^m|v filename.cha
```

7.5.5 Cross-tier COMBO

By default, COMBO excludes the header and dependent code tiers from the search and output. However, when the +t switch is used to include specified dependent tiers, 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 $INI on the %spa line, or *neat* when it is $CON on the %spa line, the following two commands could be used:

```
combo +s(want^to^$INI*)+(neat^$CON*) +g5 +t%spa +f +d sample.cha
freq +swant +sto +sneat sample.cmb
```

In this example, the +g5 option specifies that the words *want, to,* as well as the *$INI* on the %spa line may occur in any order. The +t%spa option must be added to allow the program to look at the %spa tier when searching for a match. The main tier is always searched, but dependent tiers are only searched if they are specifically included with the +t switch. 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.

7.5.6 Cluster Sequences 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 commands 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 its dependent tiers. By default, CLAN

commands work on a single cluster at a time. For COMBO, one can extend this search scope to a pair of contiguous clusters or even a sequence of several 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
```

7.5.7 Tracking Final and Initial Words

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

Then you can run FREQ on the output file.

You can use COMBO to track initial words by including the line begin %mor:, as in this example, which searches for an initial wh-word in a question.

```
combo +re +s"\%mor:^pro:wh|*^?" +t%mor +t*CHI *.cha +u
```

7.5.8 Limiting with COMBO

Often researchers want to limit their analysis to some 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 suficient and the user must 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
```

The +t*MOT switch 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. After this, comes 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*. You can then run programs like FREQ or MLU on the output.

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.

7.5.9 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.

7.5.10 Unique Options

COMBO has the following unique options:

+bN COMBO usually works on only one cluster at a time. However, when you want to look at several clusters in a row, you can use this switch. For example, +b4 would allow COMBO to search across a sequence of four clusters.

+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.

+d1 This switch outputs legal CHAT format along with line numbers and file names.

+d2 This switch outputs files names once per file only.

+d3 This 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.

+d4 Use of the +d4 switch was described in the previous section.

+d7 Search for words linked between two tiers.

+g1 COMBO can operate in either string-oriented or word-oriented mode. The default mode is word-oriented. The +g1 switch changes the mode to string-oriented. 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 search 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 string-oriented search mode 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.

+g2: Do a string-oriented search on just one word. This option is for searching for strings within each word.

+g3: Do not continue searching on a tier after first failure. This option is in cases users do not want to look for word patterns further down the tier, if the first match fails. This option is used for searches with the "not", "!", operator.

+g4: Exclude utterance delimiters from search. This will remove all utterance delimiters from the search string. It is useful, if you want to find the last word on the tier.

+g5: Make search <s1>^<s2> identical to search <s2>^<s1>. This option is used as a short cut. Normally words specified this way "word1^word2" are searched for in a specific order. This option will match for word1 and word2 regardless whether word1 precedes word2 or follows it on the tier. Otherwise user will have to specify this: (word1^word2)+(word2^word1). By default, the ^ operator means followed by, but the +g6 options turns ^ into a true AND operator. So COMBO search will succeed only if all words separated by "^" are found anywhere on the cluster tier. This also takes care of the situation when dependent tiers are not always in the same order.

+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. 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 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.

-s This switch allows you to exclude certain line from a COMBO search. It can be used in combination with the +s switch.

+t 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 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.

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.

7.6 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
```

7.6.1 Unique Options

COOCCUR has the following unique options:

+d Strip the numbers from the output data that indicate how often a certain cluster occurred.
+n Set cluster length to a certain 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.

7.7 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.

You can also use DIST to search for distances between words in the main line. For example, when looking for the word *frog* in CHILDES/Frogs/WolfHemp/08/01w8.cha, you would use this command:

```
dist +sfrog +g 01w8.cha
```

and you would get this result:

```
From file <01w8.cha>
There were 59 turns.
                  Occurrence   First    Last        Average
Word                   Count   Occurs   Occurs      Distance
-------------------------------------------------------------
frog                       8       30       51         2.6250
```

Note that the information about where the word first occurs, last occurs, and the distance between occurrences is given in terms of turns, not line numbers.

7.7.1 Unique Options

DIST has the following 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.

7.8 FREQ

One of the most powerful programs in CLAN is the FREQ program for frequency analysis. In its basic form, without special switches, 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 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 the ratio of types to tokens tends to vary with sample size.

7.8.1 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, using the +s/-s option. The FREQ program also ignores header and code tiers by default. Use the +t option if you want to include headers or coding tiers.

7.8.2 Studying Lexical Groups using the +s@file switch

The simplest use of 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 lexical groups. For example, if you want to study 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. 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 like this:

```
a
the
an
```

If you wish to use this feature to search for items on the %mor tier, then use this format:

```
m;run
m;play
m;jump
```

7.8.3 FREQ for %mor and %gra combinations

Because the %mor and %gra lines are in one-to-one alignment, you can use the +d7 switch with FREQ to search for items that have a particular characterization on both lines. For example, these commands will find words that are marked as relativizers on the %mor line and which are tagged as having the OBJ grammatical relation role on the %gra tier:

```
freq sample.cha +d7 +sm|rel +sg|OBJ
```

This feature also works when used with KWAL and COMBO.

7.8.4 Searches in Multilingual Corpora

FREQ can be used with the –l switch to track a variety of multilingual patterns. Here is a FREQ count of Spanish nouns (n|*) on the %mor tier with their corresponding words on speaker tier:

```
freq +l +s*@s:spa +sm|n +d7 *.cha
```

Here is a FREQ count of all Spanish words on %mor tier:

```
freq +l +s*@s:spa +d7 *.cha
```

Here is a FREQ count of all Spanish words on speaker tier:

```
freq +l +s*@s:spa *.cha
```

To count the frequencies of words on tiers marked with the [- spa] pre-code:

```
freq +s"[- spa]" *.cha
```

To count the number of tiers with [- spa] pre-code only:

```
freq +s"<- spa>" *.cha
```

Counting only the English tiers:

```
freq +s"<- eng>"  *.cha
```

The language pre-code has a space in it, so it is important that when you specify this language pre-code that you include the space and use quotes:

```
+s"[- spa]" +s"[- eng]" +s"<- spa>" +s"<- eng>"
```

7.8.5 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:   don’t you touch those wires .
File "0012.cha": line 1005.
*MOT:   you can’t play in those drawers .
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’t eat those [= crayons] .
File "0012.cha": line 1240.
*MOT:   no (.) you can’t eat those [= crayons] .
File "0012.cha": line 1271.
*MOT:   (be)cause you're gonna [: going 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 ?
```

7.8.6 Using Wildcards with FREQ

Some of the most powerful uses of FREQ involve the use of metacharacters including wildcards. CLAN's wildcards include

* matches any string and the results are not merged

% matches any string and the results are merged

_ matches a single word

Wildcards are particularly useful when you want to analyze the frequencies for various codes that you have entered in 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 quotation marks to guarantee that wildcards will be correctly interpreted. For Macintosh and Windows, the quotes are the best way of guaranteeing that a string is correctly interpreted. On Unix, only single quotes can be used. In Unix, single quotes are necessary when the search string contains a $, |, or > 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 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 to make sure FREQ sees these headers. The command is:

```
freq +y +s"\**:" *.cha
```

7.8.7 FREQ for the %mor line

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 rigid syntax of the %mor tier. To see the available options here, you can type "freq +sm" on the command line and you will see the following information:

```
Morphosyntactic markers specify the nature of the following string
# prefix marker
| part-of-speech marker
; stem of the word marker
- suffix marker
& nonconcatenated morpheme marker
= English translation for the stem marker
```

```
@ error word preceding [: ...] code marker on the main line
* error code inside [* ...] code marker on the main line
and then optionally - or + and followed by one of these:
      *        find any match
      %        erase any match
      string   find "string"
o erase all other elements not specified by user
o%  erase all other elements
o~  erase postclitic element, if present
o$ erase preclitic element, if present,separates alternative elements

Postclitic AND Preclitic exception:
  Find postclitics with specific Morphosyntactic marker example:
        |*,~|*   OR   ;*,~;*
  Find preclitic with specific Morphosyntactic marker example:
        |*,$|*   OR   ;*,$;*
     *        find any match
     string   find "string"
For example:
find all stems and erase all other markers:
    +sm;*,o%
find all stems and erase all other markers including all postclitics,
if present:
    +sm;*,o%,o~
find all stems of all "adv" and erase all other markers:
    +sm;*,|adv,o%
find all forms of "be" verb:
    +sm;be
find all stems and parts-of-speech and erase other markers:
    +sm;*,|*,o%
find only stems and parts-of-speech that have suffixes and eras other
markers:
    +sm;*,|*,-*,o%
find all stems, parts-of-speech and distinguish those with suffix
and erase other markers:
    +sm;*,|-*,-+*,o-%
find only stems and parts-of-speech that do not have suffixes and
erase other markers:
    -sm-* +sm;*,|*,o%
find only noun words with "poss" parts-of-speech postclitic:
    +sm|n,|n:*,~|poss
find all noun words and show postclitics, if they have any:
    +sm|n,|n:* +r4
find all noun words and erase postclitics, if they have any:
    +sm|n,|n:*,o~
```

If you are using an include file with the +s@filename option, the format of this file is:

```
m;firstword
m;anotherword
etc.
```

In other words, the format of the include file is the same as the format of the +sm options, minus the +s. When using the +sm switch with FREQ, you must also include the +t%mor switch to instruct FREQ to include searches of the %mor tier.

7.8.8 Errors for morphological codes

FREQ allows you to locate all the errors that correspond to specified morphological codes. You can do this by using the +sm switch described in the previous section. For example, to find all PASTs with all errors, you can use this command:

```
freq  +u +t*CHI +sm-PAST,** +sm&PAST,** *cha
```

The output from this will have this form:

```
From file <adler01a.cha>
Speaker: *PAR:
  1 v|deliver-PAST@dəlivd@u*n:k
```

This output shows first the frequency, then the code from the %mor line and then the error as coded on the main line. To find all PASTs with only "neg" errors, as in "word [* neg]", you can use this command:

```
freq  +u +t*CHI +sm-PAST,*neg +sm&PAST,*neg *cha
```

7.8.9 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
```

7.8.10 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 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 main speaker tier, you will be able to choose the dependent tiers of only that speaker. Each type of tier must be specifically selected 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 +s switch 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
```

7.8.11 Creating Crosstabulations in FREQ

You are use FREQ to create Excel output with crosstabulation between variables on dependent tiers. For example, if you have two coding tiers called %xarg and %xsem, you can crosstabulate using these commands:

```
freq +d8 +t%xarg +t%xsem +d2 sample.cha
freq +d8 +t%xarg +t%xsem +c5 +d2 sample.cha
freq +d8 +t%xarg +t%xsem sample.cha
freq +d8 +t%xarg +t%xsem +c5 sample.cha
```

If you want to get results across all input files, then:

```
freq +d8 +t%xarg +t%xsem +d2 +u sample.cha
freq +d8 +t%xarg +t%xsem +c5 +d2 +u sample.cha
```

7.8.12 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 +sm;*,o% sample.mor.pst
```

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

7.8.13 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. To isolate shared and unique words, you can go through three steps. To illustrate these steps, we will use the sample.cha file.

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. 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.

7.8.14 Grammatical Complexity Analysis through FREQ

Systems such as LARSP work to compute grammatical complexity from embeddings in a parse tree structure. However, this type of analysis must be done almost completely by hand and eye. One can compute many of these same relations automatically by counting the occurrences in a transcript of those grammatical relations (GRs) that mark syntactic embeddings. The relevant GRs, as described in Part 3 of this manual, are:

CSUBJ: the finite clausal subject of another clause
COMP: the clausal complement of a verb
CPRED: a full clause that serves as the predicate nominal of verbs
CPOBJ: a full clause that serves as the object of a preposition
COBJ: a full clause that serves as the direct object
CJCT: a finite clause that attaches to a verb, adjective, or adverb
XJCT: a non-finite clause that attaches to a verb, adjective, or adverb
NJCT: the head of a complex NP with a PP attached as an adjunct of a noun. The inclusion of this GR is optional.
CMOD: a finite clause that is a nominal modifier or complement
XMOD: a non-finite clause that is a nominal modifier or complement.

To pull all these out of a transcript or group of transcripts, one can use this command:

```
freq +sg|CSUBJ +sg|COMP +sg|CPRED +sg|CPOBJ +sg|COBJ +sg|CJCT
+sg|XJCT +sg|NJCT +sg|CMOD +sg|XMOD +d2 +t*PAR *.cha
```

The result is a stat.frq.xls file that you can open in Excel. It has everything you need to compute the index for each of the input files, except for the number of tokens in the files. To get that, you can run this command:

```
freq +t%gra +t*PAR +s% *.cha
```

You can then cut and paste those numbers into the first spreadsheet into a column called TokenAllGR. Then you create another Excel column, which we can call TokenComplexGRs, that sums all the frequencies of the various complexity GRs. Finally, you create a third column that divides TokenComplexGR by TokenAllGR, and that is your complexity index. Thanks to Kimberly Mueller for formulating this procedure. Using her test files, this procedure spotted 74 embeddings. Of these two were false alarms and there was one miss. So, overall accuracy of this procedure is at about 95% which compares favorably with results from human coders.

7.8.15 Unique Options

FREQ has the following unique options:

+bN This option calculates the lexical diversity of a sample using the Moving Average Type-Token Ratio (MATTR). This index is based on a moving window that computes TTRs for each successive window of fixed length (N). Initially, a window length is selected (e.g., 10 words) and the TTR for words 1-10 is estimated. Then, the TTR is estimated for words 2-11, then 3-12, and so on to the end of the text. For the final score, the estimated TTRs are averaged.

+c find capitalized words only

+c1 find words with upper case letters in the middle only

+c2 find matches for every string specified by +s option (default: only first match is counted)

+c3 find multi-word groups anywhere and in any order on a tier

+c4 find match only if tier consists solely of all words in multi-word group, even a single word

+c5 when combined with +d7 option it will reverse tier's priority (default: first tier is on top)

+c6 count only repeat segments when searching for words with ↫

+c7 for multi-word groups, this searches without expanding wildcards. For example, in combination with +s"set *" it will give the count of words occuring combination with "set".

+dCN output only words used by <, <=, =, => or > than N percent of speakers

+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 form that can be opened directly in Excel. To do this, you must include information about the speaker roles you wish to include in the output spreadsheet.
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. Word frequencies are not placed into the output. You do not need to use the +f option with +d2 or +d3, since this is assumed.

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

+d5 This switch will output all words you are searching for, including those that occur with zero frequency. This could happen, for example, if you use the +s switch to search for a specific word and that word does not occur in the transcript. This switch can be combined with other +d switches.

+d6 When used for searches on the main line, this switch will output matched forms with a separate tabulation of replaced forms, errors, partial omissions, and full forms, as in this example for "going" as the target word:

```
17 going
    3 going
        11 gonna [: going to]
          2 goin(g)
    1 go [: going] [* 0ing]
```

This switch can also be used on the %mor line in a form such as this:

```
freq +d6 adler01a.cha +sm|n*,o%
```

will produce separate counts for all instantiations of a given part of speech, organized by part of speech. For example, the output for n:gerund would be:

```
2 n:gerund
      1 n:gerund|go-PRESP
      1 n:gerund|look-PRESP
```

+d7 This command links forms on a "source" tier with their corresponding words on a "target" tier, yielding output such as this:

```
12 pro|you
       8 you
       4 you're
```

In this example, and by default, the first line gives the form on the %mor line as the source tier, and the following lines give the corresponding main line or "target" words. If you add the name of a tier, such as %gra, then that becomes the source. This switch expects that the items on the two tiers be in one-to-one correspondence. If you want to switch the display so that the target becomes the source, you can add the +c5 switch. You can also specify a match between two dependent tiers, as in this example:

```
freq +d7 +sm|cop +sg|ROOT +t%gra +t%mor t.cha
```

+d8 outputs words and frequencies of cross tabulation of one dependent tier with another

+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.

+o1 sort output by reverse concordance

+o2 sort by reverse concordance of first word; non-CHAT, preserve the whole line

+o3 By default, FREQ tabulates separate frequencies for each speaker. To see the combined results across all speakers, use this switch.

+pS add S to word delimiters. (+p_ will break New_York into two words)

FREQ also uses several options that are shared with other commands, such as +f, +k, +l, +y, +r, +s, +u, +x, +z, and others. 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.

7.8.16 Further Illustrations

Davida Fromm has created this list of further examples of the use of FREQ:

1. If you want the frequency of all words used by Participants on the %mor line in descending order of frequency, file by file, with their part of speech label, use:

```
freq +t%mor +t*PAR -t* +o *.cha
```

If you want to merge results across files, add +u to the above command. If you want to exclude unintelligible words and neologisms, add -sm|neo,|unk" to the above command.

2. If you want to do this on the speaker line and exclude unintelligible words and neologisms, use:

```
freq +t*PAR +o -s"xx" -s"<*\* n:uk*>" *.cha
```

If you want to get information about error productions, add +d6 to the above command.

If you want to send the results to a file instead of having them appear on the computer screen, add +f to the above command.

3. If you want the frequency of all words from the %mor line in descending order, by stems, with information on parts of speech, bound morphemes, and error codes, use:

```
freq +t*PAR +d6 +o +s"@r-*,o-%" *.cha
```

4. If you want a list and frequency count of all prefixes used by Participants in descending order of frequency, merged across files in the folder, use:

```
freq +t*PAR +o +s"@r-*,#-*,o-%" +u *.cha
```

If you want the %mor line printed out with that information, use +d.

5. To get a frequency count of Participant word-level errors (see the Error Coding sheet at the website for a description of these error codes), file by file, a basic command is:

```
freq +s"\[\* *\]"  +t*PAR  *.cha
```

If you want to include errors that were repeated and revised, add +r6 to the above command.

6. If you want to specify which errors you want listed and counted, you can list them as in the following command. Remember to add +r6 to the command if you want word-level errors within repetitions and retracings (e.g., *s:r, *s:r-rep, *s:r-ret).

```
freq +s"[\* s*]" +s"[\* p*]" +s"[\* n*]" +s"[\* d*]" +s"[\* m*]"
+s"[\* f*]" +t*PAR *.cha
```

7. Alternatively, you can create a CLAN cut file of all error types and use that instead. Put the cut file in the same folder as the files you are analyzing and use:

```
freq +s@error.cut +t*PAR *.cha
```

Remember to add +r6 to the command if you want to include errors within repetitions and retracings.

If you want to create a list with frequencies of each error and the CHAT transcript line that includes that error, add +d to the above command line.

If you want a tally and list of the actual errors with the intended target word, add +d6 to the above command line.

If you want the context (CHAT transcript line) as well, add +d to the above command line instead of +d6.

8. If you want the data to be in a file that can be opened in Excel, use:

```
freq +s"[\* s:r]" +s"[\* s:ur]" +s"[\* s:uk]" +s"[\* s:per]" +s"[\*
p:w]" +s"[\* p:n]" +s"[\* p:m]" +t*PAR +d2 +fS +2 *.cha
```

Triple click on the line with the filename at the end of the CLAN output to open the Excel file. The Excel file itself will be in the folder you put as your working directory.

9. To get a frequency count of Participant errors at the sentence level (see the Error Coding sheet at the website for a description of these error codes), a basic command is:

```
freq +s"<+ >" +t*PAR *.cha
```

If you want a certain type of sentence error, for example jargon, use <+ jar> inside the

quotation marks.

10. If you want to see all error productions associated with a target word, for example, Cinderella, use:

```
freq +s"<: Cinderella>" *.cha
```

11. To list all parts of speech that occur in the files, merged across all the files, with their corresponding frequencies, in descending order of frequency, use:

```
freq +t*PAR +d5 +o +sm|*,o% +u *.cha
```

12. If you want to list and count the frequency of all verb forms, stems ony, merged across files in a folder, use:

```
freq +t*PAR +sm;*,|v*,|aux*,|part*,|mod*,|cop*,o% +u *.cha
```

13. If you want the total number of nouns (of all types), stems only, merged across files in a folder, use:

```
freq +t*PAR +d5 +o +sm|n,|n:*,o-% +u *.cha
```

14. If you want to count and list the nouns, stems only, merged across files, use:

```
freq +t*PAR +d5 +o +sm;*,|n,|n:*,o% +u *.cha
```

15. If you want word-level errors and other part of speech and bound morpheme info about the noun, merged across files, use:

```
freq +t*PAR +d6 +o +sm;*,|n,|n:*,o% +u *.cha
```

16. If you want to list and count the frequency (in descending order) of all prepositions used by the Participant merged across files in a folder, use:

```
freq +t*PAR +d5 +o +sm;*,|prep*,o% +u *.cha
```

If you want to see the lines in which they're used, use +d instead of +d5.

17. If you want to list and count the frequency (in descending order) of all adverbs used by the Participant, by stems, merged across files in a folder, use:

```
freq +t*PAR +d5 +o +sm;*,|adv*,o% +u *gem.cex
```

18. If you want to list and count the frequency (in descending order) of all adjectives used by the Participant, by stems, merged across files in a folder, use:

```
freq +t*PAR +d5 +o +sm;*,|adj*,o% +u *gem.cex
```

7.9 FREQMERG

If you have collected many 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.

7.10 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+chalkinitial =  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. To study occurrences in second position, we must use the +d switch as in:

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

7.10.1 Unique Options

FREQPOS has the following 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 commands. 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 word with FREQPOS, you should use the +g switch rather than the +s switch.

7.11 GEM

Researchers use gem markers for different purposes. In some CHILDES corpora, they are used to mark the dates or numbers of diary entries. In studies of narratives and book reading, they are used to mark page numbers. In tasks with object and picture description, they may indicate the number or name of the picture. In some corpora, they are used just to enter descriptive remarks.

One important and interesting use of gems is to facilitate later retrieval and analysis. For example, some studies with children make use of a fixed sets of activities such as MotherPlay, book reading, and story telling. For these gems, it can be useful to compare

similar activities across transcripts. To support this, we have entered the possible gems in a corpus that uses gems in this way into the TalkBankDB facility in a pulldown menu. Descriptions of the gems used in a corpus can be found in the homepage for that corpus.

The GEM program is designed to allow you to pull out 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 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.

7.11.1 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 specific 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 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 specific 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
```

7.11.2 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 segments of the file for analysis. When you do this, you will want to use the +d and +f switches, so that the output is in CHAT format foir analysis by further commands.

```
gem +sreading +d +f sample2.cha
```

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.

7.11.3 Unique Options

GEM has the following 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 the tag words specified with +s switches must appear on the @Bg: header line 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 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 @G: 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.

7.12 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
```

7.12.1 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 the tag words specified with +s switches must appear on the @Bg: header line 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 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.

7.13 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.

7.14 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.

7.14.17 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 treat the metacharacters as literal, 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.

7.14.2 Unique Options

KEYMAP has the following unique options:

+b This is the beginning specification symbol.
+o In addition to tracking following codes, this option also tracks preceding codes.
+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
```

7.15 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. 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 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.

7.15.1 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, 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 +f
flo *.kwal.cex
```

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
```

7.15.2 KWAL for breaking up files by code type

In a variation of the procedures in the last section, you may want to break out a certain type of utterance from a file for further analysis or even break up a file into several segments. For example, if you have used a code such as $A on the %cod tier and want to contrast sentence with and without that code, you can pull out all the utterances with $A using this command:

```
kwal +d +o@ +t% +s"$A" +f$A filenames
```

This will produce the $A.cex output file. Next, you can créate a file with the utterances that do not have $A using this command:

```
kwal +d +o@ +t% -s"$A" +fno$A filenames
```

This will produce the no$A.cex output file. Then you can run commands on the output files separately.

7.15.3 KWAL for %mor and %gra combinations

You can use the +d7 switch in COMBO, FREQ, and KWAL to search for matches between the %gra and %mor lines, because these two lines are in one-to-one correspondence. Here is an example of a command to search for cases in which the SUBJ on the %gra line is pro:per on the %mor line. In this command, the letter “g” after +s refers

to the %gra tier and the letter "m" after +s refers to the %mor tier.

```
kwal +d7 +sg|SUBJ +sm|pro:per filename.cha
```

You can also search for cases in which the SUBJ on the %gra line is pro:per|you on the %mor line. In this command, the letter "g" after the +s refers to the %gra tier and the letter "m" refers to the %mor tier.

```
kwal +d7 +sg|SUBJ +sm|pro:per,;you filename.cha
```

7.15.4 KWAL with signs and speech

When participants are using mostly nonspeech items coded on the %sin dependent tiers but a few speech items on the main tier (coded as 0 items), then it is sometimes interesting to track utterances in which speech is either combined or not combined with gesture. The +c switch can help in this analysis by selecting out utterances that have a small number of words on the main line, but extensive coding on the %sin line. Here are some sample command sequences for this type of work:

1. Gesture only (at least one g: on the %sin tier, no words on the main tier and no s: on the %sin tier).

   ```
   kwal *.cha +t%sin +c0 -ss:* +d +fno-s
   kwal *.no-s.cex +t%sin +sg:* +d +fgesture_only
   ```

2. Speech only (included at least one word on the main tier, but no g: or s: on the %sin tier):

   ```
   kwal *.cha +t%sin +c1 -ss:* -sg:* +d +fspeech_only
   ```

3. Sign only (included at least one s: on the %sin tier, but no words on the main line and no g: on the %sin tier):

   ```
   kwal *.cha +t%sin +c0 -sg:* +d +fno-g
   kwal *.no-g.cex +t%sin +ss:* +d +fsign_only
   ```

4. Gesture + speech only (included at least one g: on the %sin tier and at least one word on the main tier, but no s: on the %sin tier):

   ```
   kwal *.cha +t%sin +c1 -ss:* +d +fno-s
   kwal *.no-s.cex +t%sin +sg:* +d +fgesture+speech
   ```

5. Gesture + sign only (included at least one g: and one s: on the %sin tier but no words on the main tier):

   ```
   kwal *.cha +t%sin +c0 +ss:* +d +fsign_only
   kwal *.sign_only.cex +t%sin +sg:* +d +fgesture+sign_only
   ```

6. Gesture + speech + sign (included at least one g: and one s: on the %sin tier and at least one word on the main tier):

   ```
   kwal *.cha +t%sin +c1 +ss:* +d +fsign_only
   kwal *.sign_only.cex +t%sin +sg:* +d +fgesture+speech+sign
   ```

7.15.5 Unique Options

KWAL has the following 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 in these formats:
+d : outputs legal CHAT format
+d1: outputs legal CHAT format plus file names and line numbers
+d2: outputs file names once per file only
+d3: outputs ONLY matched items
+d30: outputs ONLY matched items without any defaults removed.
The +d30 and the +d3 switches can be combined.
+d99: convert "word [x 2]" to "word [/] word" and so on
+d4: outputs for Excel
+d40: outputs for Excel, repeating the same tier for every keyword match
+d7: compares items across dependent tiers, as in this example:
kwal +d7 +s@|-cop +sROOT +t%gra +t%mor t.cha

+/-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. 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@ID -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.

+**xCN**include only utterances which are C (>, <, =) than N items (w, c, m), "+x=0w" for zero words

+xS specify items to include in above count (Example: +xxxx +xyyy)

-xS specify items to exclude from above count

7.16 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 the words or utterances of a certain length. For example, the following command will locate all 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%
```

7.16.1 Unique Options

+a If you have elected to use the +c switch, you can use the +a 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 +a +xw sample.cha
```

+b You can use this switch to either include or exclude specified 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 can use the following

command:

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

Then you would run the output through 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.

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

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

7.16.2 Unique Options

MAXWD has the following unique options:

+a : consider ONLY unique length utterances/words

+cN: display N longest utterances/words

+gN: look for longest utterance instead of longest word

1 - number of morph; 2 - number of word; 3 - number of chars

7.17 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 ignored in this computation. Instead, the program simply looks for sequences of repeated speaker ID codes at the beginning of the main line. While the same speaker is talking, then each utterance is a part of the current turn. These computations are provided for each speaker separately. Note that none of these ratios involve morphemes on the %mor line. If you want to analyze morphemes per utterances, you should use the MLU program.

7.17.1 MLT defaults

The exact nature of the MLT calculation depends both on what the program includes and what it excludes. By default, all utterances and words are included with these default exceptions:

1. MLT excludes material followed by [/], [//], [///], [/?], and [/-]. This can be changed by using the +r6 switch or by adding any of these switches: +s+"</>" +s+"<//>"
2. The following strings are also excluded as words: 0* &* +* -* $*. Here the asterisk indicates any material following the first symbol until a delimiter.
3. The symbols xxx, yyy, and www are also excluded from the word count by default, but the utterances in which they appear are not. If you wish to also exclude these utterances, use the switches –sxxx, -syyy, and –swww.
4. Utterances with no words are included, although they can be excluded using the +a

switch.

To exclude utterances with a specific postcode, such as [+ bch], use -s"[+ bch]". Similarly, you can use +s to include lines that would otherwise be excluded. Pairs of utterances that use the +, and +. continuation codes are counted as single utterances. If a potential turn is composed only of utterances that are being excluded with the –s option, then the whole turn also disappears and the surrounding two turns become a single turn. In addition, you can use the +g option to exclude any utterance composed exclusively of a certain set of words. Because MLT's exclusions are not as extreme as those for MLU, there are often more utterances included in MLT than in MLU.

7.17.2 Unique Options

MLT has the following unique options:

+a This switch causes all utterances to be counted, even if they have no words. If you add "t" to make +at, you will only count empty utterances if they have the [+ trn] postcode.

+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 to output data in a format that can be opened in Excel, as in this command:

```
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 a given participant for a particular file. 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 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 certain words. For example, you might wish to exclude utterances composed only of *hi*, *bye*, or both 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, i.e. 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 string that specifies which utterances should be included. This switch selects whole utterances for inclusion, not individual words, because MLT is an utterance-oriented program.

7.18 MLU

The MLU program computes the mean length of utterance, which is the ratio of morphemes to utterances. By default, this program runs from a %mor line and uses that line to compute the mean length of utterance (MLU) in morphemes. However, if you do not have a %mor line in your transcript, you need to add the –t%mor switch to use it from the main line. In that case, you will be computing MLU in words, not morphemes.

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 described language growth through six stages of development for which MLU values ranged from 1.75 to 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 count 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 detailed description, corresponding to Brown's nine points.

1. 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. This is the form of the command used for MLU-100 in the KIDEVAL program.
2. The symbols xxx, yyy, and www are also excluded by default, as are the utterances in which they appear. If you wish to include the xxx forms and the utterances that contain them, then use the +sxxx option. The forms yyy and www are always excluded and cannot be included. Utterances with no words are excluded from the utterance count.
3. If you mark repetitions and retraces using the CHAT codes of [/], [//], [///], [/?], and [/-], the repeated material will be excluded from the computation automatically. This can be changed by using the +r6 switch or by adding any of these switches: +s+"</>" +s+"<//>".
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: uh um 0* &* +* -* $* where the asterisk indicates any material following the exclusion symbol. 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 *uh* by default. This is done by marking them as &-mm and &-uh.
5. 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 also 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". You can also use the special form marker @a to force treatment of an incomprehensible string as a word. This can happen when the sentential and situational context is so clear that you know that the form was a word. For example, the form xxx@a will appear on the %mor line as w|xxx and the form xxx@a$n will appear on the %mor line as n|xxx. In the place of the "n" you could place any part of speech code such as "v" or "adj". This is because the @a is translated through the code "w" as a generic "word" and the part of speech code after the $ sign is translated as the part of the speech of the incomprehensible word. These codes also apply to yyy, yyy@a, and yyy@a$n.

6. When MLU is computed from the %mor line, the compound marker is excluded as a morpheme delimiter, so this restriction is automatic. If you compute MLU from the main line, then you need to add –b+ to your command to exclude the plus as a morpheme delimiter.
7. The ampersand (&) marker for irregular morphology is not treated as a morpheme delimiter, so this restriction is automatic.
8. 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.
9. 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.

It is also possible to exclude utterances by using postcodes. By default, MLU excludes utterances marked with the specific postcode [+ mlue]. This works both for MLU as a separate program and for MLU as a part of KIDEVAL. It is also possible to make further exclusions for MLU as a separate program by using some other postcode such as [+ exc] in the form of -s"[+ exc]". However, this non-default marking will not get picked up by KIDEVAL.

The use of postcodes for exclusion 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.

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 depends on the correct morphemicization of 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.

7.18.1 Exclude files for MLU and MLT

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

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

This command looks at only those utterances spoken by the child to the mother as addressee. You can then run MLU on the output of the KWAL command.

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 can 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 following 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 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.

7.18.2 Unique Options

MLU has the following unique options:

+b You can use this switch to either include or exclude certain 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 to output data for Excel, as in this command:

```
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. 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 collection of files. In this case, all 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 certain 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. 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. 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.

7.19 MODREP

The MODREP program matches words on one tier with corresponding words on another tier. It works only on tiers where every 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 rep-

licated 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 to guarantee that the analysis of the %pho line is case-sensitive. By default, all CLAN commands except for FREQ, FREQMERGE, MORTABLE, PHONFREQ, RELY, TIMEDUR, and VOCD are case-insensitive.

7.19.1 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*
```

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.

7.19.2 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
```

7.19.3 MODREP for the %mor line

Because words on the main line stand in a one-to-one relation with words on the %mor line, 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 analyze the output using the MODREP command we used earlier.

```
combo +s"v:*^*^n:*-acc" +t%mor sample2.cha +d +f
modrep +b%mor +c*MOT +o"*acc"sample2.cmb.cex
```

The output of this program is the same as in the previous example. However, in a large input file, the addition of the COMBO filter can make the search much more restrictive and powerful.

7.19.4 Unique Options

MODREP has the following unique options:

+a sort output by descending frequency

+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 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 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.

+s The +s switch allows you to include particular symbols such as xxx or &* that are excluded by default. The -s 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.

7.20 Phon and PhonTalk

PHON is Java program that performs extensive detailed analyses of phonological patterns in transcripts linked to media. PHON was created by Greg Hedlund and Yvan Rose at Memorial University of Newfoundland. Information on how to download, install, and learn PHON is available at https://www.phon.ca/phon-manual/misc/Welcome.html. Although PHON is not a part of CLAN, the two systems use a common form of XML that makes it possible to convert CHAT files to PHON projects and also to convert PHON projects to a series of CHAT files. To do this, download Chatter from https://talkbank.org/software/chatter.html and PhonTalk from https://www.phon.ca/phon-manual/chat/phontalk.html.

When converting from CHAT to Phon, first run Chatter to make sure that your files are in good CHAT format. To do this, you make sure that the buttons the select "from CHAT to XML" are turned on. Then, you select the folder you want to convert using the *Open* in the *File* pulldown menu. Then Chatter will run on that folder and create a new folder with the name of the original plus -XML. For example, if you run on the Bliss folder, it will create Bliss-xml.

Next, you should open PhonTalk, and select the *Open* option in the *File* pulldown menu. The program will run on each file and will list each as "Finished" once it is processed. By default, the output in the case of the Bliss corpus would be Bliss-xml-phon. This is then a Phon project which you can analyze using Phon. You may want to rename this output to remove the "xml" in the folder name, as in Bliss-phon.

To convert back to CHAT, you start PhonTalk and select Bliss-phon by choosing *Open* from the *File* menu. This will create a CHAT XML version of the Phon files. Then you run Chatter to convert from CHAT XML to standard CHAT, this time selecting the radio buttons for "XML to CHAT".

7.21 RELY

This program has five functions: (1) to examine agreement between two coders, (2) to compute Cohen's kappa, (3) to evaluate correctness of a student coder against a master document, (4) to check overall match between two transcripts, and (5) to combine codes into a single file.

First, we will consider the function of checking reliability between two coders. 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 or %spa line and then use the RELY program to spot matches and mismatches. To create an example, you could copy the sample.cha file in CLAN's /examples folder to a file called samplea.cha file and change one code in the samplea.cha file. In samplea.cha, change the code for the first utterance from "$INI:sel:in" to "$INI:sel:gone". Then enter the command:

```
rely sample.cha samplea.cha
```

The output in sample.rely.cex file will report the coding disagreements and you can triple-click on the lines that give the line numbers, and they will open to the point of the mismatch in each file. If you add +t%spa to this command, you will get a fuller report. Counterintuitively, if you only care about mismatches on the %spa line, you can leave out the +t%spa switch. This will work in the same way, if your coding tier name is, for example, %cod or something else.

By default, RELY examines all main and dependent tiers. In the example discussed so far, the two transcripts are identical except for any differences that would appears on the coding tier. However, in some other cases, there could also be differences on the main line or elsewhere. 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. Then, you will need to use the +t switch to pick out a line or speaker to include, while ignoring all the others. If the command is:

```
rely +c sample.cha samplea.cha +t%spa -t*
```

then the program will only report mismatches on the %spa tier. Note that, in addition to adding the +c switch, you must also add the -t* switch to make sure the RELY also ignores the main line in addition to ignoring other depending tiers. If you further add a +t*CHI switch, it will only report mismatches for selected tier for the Target_Child. However, if you use the +c switch with no additional +t inclusions, RELY will report unmatched tiers, but not the actual mismatches. In the sample.rely.cex output, you can triple-click on lines with the line numbers given and CLAN will open to that place in the original file.

A second function of RELY is to compute Cohen's kappa by adding the +dN switch. For this analysis, you need to specify the number of possible categories being checked. Here is an example of computing kappa for %spa lines with three possible categories:

```
rely +c +d3 sample.cha samplea.cha +t%spa -t*
```

The third function of the RELY command is to evaluate the correctness of a student transcript against a master transcript. The student is given a version of the complete master transcript without the coding tier and the goal is to add codes. In this case, it is assumed that the coding in the master transcript is fully correct. The question is whether the student's codes are correct. To specify that the first file given is the master, you use the +dm1 switch.

```
rely +dm1 sample.cha samplea.cha +t%spa
```

The output from this command provides two counts: precision and accuracy. These are the two dimensions used in fields such as computational linguistics to evaluate correctness. Precision is the percentage of fields in the master file that are matched by the student. Accuracy is the percentage of fields in the student file that are matched to the master. A failure in precision occurs when the student misses a code in the master file. A failure in

accuracy occurs when the student inserts a code that is not found in the master. If a researcher is interested in going beyond these two scores, they can also compute the F score which is the harmonic mean of precision and accuracy.

The fourth function of the RELY command is to estimate the overall match between two transcripts on the main line. It is very difficult to define this type of comparison precisely. Instead, RELY uses a rough-and-ready "bag of words" comparison method that simply looks at the overall match of the main line items in the two versions. The command for this type of analysis adds the +d switch, and the output is the percentage of overall overlap.

```
rely +d sample.cha samplea.cha
```

The fifth function of the RELY command is to allow multiple coders to add a series of dependent tiers to a master file. The master file is the first one given in the command line. The lines of the master file should remain untouched and the coder of the second file should only be adding information on a single additional dependent tier. This function is accessed through the +a switch, which tells the program the name of the coding line (given by the +t switch) from the secondary file that is to be added to the master file, as in

```
rely +a +t%spa +t@ sample.cha samplea.cha
```

If, by mistake, some changes were made to the other coding lines, the output will ignore the mismatches, keeping what is in the master file only. To get a full file merger, you need to add the +t@ switch to include the header tiers.

It is important to understand the detailed workings of comparison with the +a switch. When used with +a and +t%cod, RELY looks at the first speaker tier in master and coder files and, if they match, then it looks for the dependent tier in coder file that was specified with +t%cod option. If it finds a %cod tier in coder file, then it looks in the master file under the corresponding speaker tier to see if the master file already has a %cod tier. If it does (and it really shouldn't), then the error message "** Duplicate tier found around lines:" is given and user must choose whether the master file tier or the coder file tier should be added to master file. If master file does not already have %cod tier, then the %cod tier from coder file is add to all the other dependent tiers in the master file under the corresponding speaker tier.

RELY +a will also report an error, if it finds some other dependent tier, such as %com in the coder file that is not in the master. In that case, it will report a message "** Unmatched tiers found around lines:" to inform user that there are tiers in the coder file that are not in the master file and which have not been specified by the user to be added with +t%com option. This message is just an FYI.

RELY +a will only add tiers from coder file that are missing from master file and are specified with +t option. At the same time, it will report if there are some other tiers in coder file that were not specified with +t option that are also missing from master file. In other words, the coder file must be a subset of master file with only extra tiers that users would want to add to master file. However, possibly not all those tiers are supposed to be added and that is what the +t option is for.

If you want to conduct multiple runs with RELY, looking at different speakers and different coding lines using the +c and +t switches, then you may also want to use the +2 switch to create differently named files from each run of RELY.

7.21.1 Unique Options

RELY has the following unique options:

+a	Add tiers from second file to the master file.
+b	Include media bullets in string comparison between first and master files and show differences if there are any.
+c	Only check data on tiers selected with +t.
+d	Compute percentage agreement. By default, this is based only on the main line. To compute percentage agreement on a dependent tier, such as %cod, you should add the –t* switch to exclude the main line and then use +t%cod to include just this dependent tier.
+dmN:	Compute student correctness. (+dm1 - first file is control, +dm2 second file is control)
+dN	Compute Cohen's kappa coefficient, where N is the number of categories.
+m	Merge files and place error flags inside output file.

7.22 SCRIPT

The SCRIPT command is useful if fixed scripts are used in clinical research. It will compare a participant's performance to that of the model. To run SCRIPT, you must first prepare a Model Script and a Participant's Script.

7.22.1 The Model Script

1. Transcribe the model script in CHAT format. It is not necessary to have a media file or do any linking on this model script. If sample duration is important, however, include @ Time Duration as a header.
2. Run CHECK (using esc-L) to verify that CHAT format is accurate and run mor +xl *.cha to make sure the words in the file are all recognizable. Run MOR, POST, and CHECK. Put this file in your CLAN lib folder in the folder with the Participant's files you will be comparing to it.

7.22.2 The Participant's Script

Transcribe the Participant's script production in CHAT format and link it to the media. The analysis compares the model and Participant transcript on a line-by-line basis, so it is necessary for the Participant transcript lines to match those of the model.

1. If the Participant skips a line of script, enter 0 for that line (*PAR: 0.).
2. If the Participant makes comments (e.g., "I don't know", "oh no") in the middle of an utterance, put those words inside angle brackets with [//] afterward so they will not be counted as extra words in the analysis.
3. Error productions that are followed by target replacements must be judged to be either: 1) close approximations of the target; or 2) not close approximations. CLAN will include close approximation errors as correct matches to the target word when the SCRIPT command is run in its default mode. This can be easily modified, if such an analysis is not desired (see Variations below).

4. Utterances with the [+ exc] code are ignored.
5. If most listeners would be able to figure out what the person meant, the error should be considered a close approximation and should be followed by the target replacement word like this: error [: target]. Usually this means the error production is a close semantic synonym (gifts for giftware), a lexical error (e.g., have for had), or a production with 1 or 2 phonemic errors (e.g., /əʃeʒə/ for aphasia. If in doubt, assume it is not a close approximation.
6. If an error is judged not to be a close approximation and most listeners would be unable to know what the speaker meant (e.g., May for March, say for attend, reading for write), transcribe the error word with the target replacement like this: error [:: target].

Once the transcription is complete, run CHECK (using esc-L) to verify that CHAT format is accurate and run mor +xl *.cha to make sure the words in the file are all recognizable. Run MOR, POST, and CHECK.

7.22.3 Running SCRIPT

Once you have prepared the two scripts, this command will compare a model script file (which we will call model.cha) to a participant's script file (which we will call participant.cha):

```
script +t*PAR +smodel.cha participant.cha
```

The output will include an .xls file and a .cex file. The .xls file provides the following information for both the model script and the Participant's script production: TIMDUR and # words produced. It provides the following information on the Participant's script only: # words correct, % words correct, # words omitted, % words omitted, # words added, # recognizable errors, # unrecognizable errors, # utterances with xxx, # utterances with 0. Unrecognizable errors are those transcribed as xxx or coded as unknown neologistic or semantic errors ([* n:uk] and [* s:uk]. (See the chapter on Error Coding in the CHAT manual.) All other errors include target replacements and are considered recognizable.

The .cex file provides the following information for the Participant's script production: list of omitted words (with part of speech and bound morpheme), list of added words, and list of errors (error production, intended word if known, error code and frequency info).

7.22.4 Variations

If you want to produce output for all the CHAT files in a folder you would use this command:

```
script +t*PAR +smodel.cha *.cha +u
```

The +u switch will list the results for each CHAT file instead of individual .cex and .xls files.

The default mode for this command is to INCLUDE target replacements for errors judged to be close approximations (lake [: like] [* p:w]) and EXCLUDE revisions and retracings (anything coded with [/] or [//]). Both of those defaults can be changed by adding switches to the command line:

+r5 excludes target replacements

+r6 includes repetitions and revisions

7.22.5 Unique Options

MAXWD has the following unique options:

+e : count error codes in retraces or repeats. (default: don't count)

+sF: specify template script file F

7.23 TIMEDUR

The TIMEDUR program computes the duration of the pauses between speakers and the duration of overlaps. This program requires sound bullets at the ends of utterances or lines 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 basic output format of TIMEDUR gives a profile of durations for all speakers through the whole file. For a more succinct summary of durations for a given speaker, use a command with the +t switch, such as:

```
timedur +d1 +d +t*PAR *.cha
```

This command creates a summary of time durations across files for just PAR. In effect, it treats the +u switch as the default.

+d outputs default results in SPREADSHEET format
+d1 outputs ratio of words and utterances over time duration
+d10 outputs above, +d1, results in SPREADSHEET format

7.24 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. The complete description of VOCD can be found in: Malvern, D., Richards, B., Chipere, N., & Purán, P. (2004). *Lexical diversity and language development*. New York: Palgrave Macmillan.

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. This problem has distorted research findings. 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, Chipere, & Purán, 2004).

7.24.1 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 (Brainerd & Pressley, 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 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 with the same general and appropriate shape, with different values for the parameter D distinguishing different members of this family (Malvern & Richards, 1997a). In the model, D itself is used directly as an index of lexical diversity.

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, 1997a, 1997b; Malvern et al., 2004; Richards & Malvern, 1996) 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) (Richards & Malvern, 1996, pp. 35-38), correlated well with other well validated measures of language. These five corpora also provide useful indications of the scale for D.

7.24.2 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 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, 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 approximates 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. Based on 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.

7.24.3 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.

7.24.4 VOCD Running and Output

By default, VOCD runs from the %mor line. This is also true for the form of VOCD that runs in KIDEVAL. This is necessary, because the %mor line provides the base form of each word or lemma, which is the best way to compute lexical diversity. Also, the %mor line excludes repetitions.

If you do not have a %mor line in your transcript(s), then you should use the +o switch to run VOCD from the main line instead. Of course, running in this mode will lead to results based on surface forms, rather than lemmas.

Because VOCD runs by default off the %mor line, you should use the form of the +/-s switch that conforms with the syntax of the %mor line. This form uses +sm, rather than something like +s"n|*". You can best understand the syntax of +sm by typing:

```
vocd +sm
```

In the CLAN output window, you will then see a description of the various components of the +sm switch, along with example usages. When you use the +sm version of the search string, you then make sure that VOCD is running from the %mor line.

To illustrate the functioning of VOCD, we can use a command that examines the child's output in the file 68.cha in /examples/transcripts/ne32. The command for doing this is:

```
vocd +t*CHI +sm;*,o% 68.cha
```

To also exclude affixes and neologisms (unintelligible words are already excluded from this analysis), use:

```
vocd +t*CHI +sm;*,o% -sm|neo +f *.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. Three tables that each show 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.

4. 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 three 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    0.7846    0.059     50.003
  36      100    0.7739    0.055     47.677
  37      100    0.7786    0.057     50.673
  38      100    0.7626    0.063     46.554
  39      100    0.7700    0.057     50.268
  40      100    0.7550    0.056     46.533
  41      100    0.7590    0.064     49.011
  42      100    0.7626    0.055     51.450
  43      100    0.7521    0.057     49.056
  44      100    0.7423    0.052     47.032
  45      100    0.7442    0.053     48.722
  46      100    0.7391    0.059     48.167
  47      100    0.7334    0.052     47.413
  48      100    0.7304    0.053     47.496
  49      100    0.7224    0.060     46.072
  50      100    0.7268    0.058     48.338

D: average = 48.404; std dev. = 1.541
D_optimum     <48.34; min least sq val = 0.000>

VOCD RESULTS SUMMARY
====================
   Types,Tokens,TTR:  <121,388,0.311856>
  D_optimum  values:  <49.61, 47.22, 48.34>
  D_optimum average:  48.39
```

7.24.5 Unique Options

VOCD has the following unique options:

+d outputs a list of utterances processed and number of types, tokens and TTR, but does not calculate D.
+d1: outputs only VOCD results summary on one line
+d2: outputs only type/token information
+d3: outputs only type/token information in Excel format
+b various methods for controlling sampling
+b0: D_optimum - use split half; even.
+b1: D_optimum - use split half; odd.
+bsN: D_optimum - size N of starting sample (default 35)
+blN: D_optimum - size N of largest sample (default 50)
+biN: D_optimum - size N of increments (default 1)
+bnN: D_optimum - the N number of samples (default 100)
+br: D_optimum - random sampling with replacement (default: noreplacement)
+be: D_optimum - use sequential sampling
+g: Calls up the limiting relative diversity (LRD) sub-routine to compare the relative

diversity of two different word classes, the numerator and the denominator, coded on the %mor tier. This procedure extracts the words to be included from the %mor tier where the word classes are coded. The speaker, the %mor tier and the file name are specified in the usual way, plus the +g switch to invoke the subroutine. The following would compare verb and noun diversity and limit the analysis to word stems:

```
vocd +t*CHI +t%mor +gn"m|v,;*,o%" +gd"m|n,;*,o%" filename
```

The first word class entered will be the numerator and the second will

be the denominator.

+gnS: compute "limiting type-type ratio" S=NUMERATOR

-gnS:compute "limiting type-type ratio" S=NUMERATOR

+gdS: compute "limiting type-type ratio" S=DENOMINATOR

-gdS:compute "limiting type-type ratio" S=DENOMINATOR

7.25 WDLEN

The WDLEN program tabulates the lengths of words, utterances, and turns. The basic command is:

```
wdlen sample.cha
```

The output from running this on the sample.cha file will be as displayed here:

```
Number of words of each length in characters
lengths:    1    2    3    4    5    6    7    8    9   Mean
*CHI:       0    0    0    4    4    0    0    0    0   4.50
*MOT:       2    4   11   11   11    0    0    0    2   4.00
-------
Number of utterances of each of these lengths in words
lengths:    1    2    3    4    5    6    7    8    9   10   Mean
*CHI:       3    1    1    0    0    0    0    0    0    0   1.60
*MOT:       0    1    1    1    2    2    0    0    0    1   3.77
-------
Number of single turns of each of these lengths in utterances
lengths:    1    2   Mean
*CHI:       0    0   0.00
*MOT:       0    1   2.00
-------
Number of single turns of each of these lengths in words
lengths:    1    2    3    4    5    6    7    8    9   10   11
Mean
*CHI:       0    0    0    0    0    0    0    0    0    0    0
0.00
*MOT:       0    0    0    0    0    0    0    0    0    0    1
11.00
-------
Number of words of each of these morpheme lengths
lengths:    1    2   Mean
*CHI:       7    1   1.12
*MOT:      35    7   1.16
-------
Number of utterances of each in morpheme length:
```

```
          1     2     3     4     5     6     7     8     9    10    11    12
Mean
*CHI:     3     0     2     0     0     0     0     0     0     0     0     0
1.80
*MOT:     0     0     2     0     1     2     2     0     0     0     0     1
4.46
```

The first four analyses are computed from the main line. For these, the default value of the +r5 switch is shifted to "no replacement" so that word length is judged from the actual surface word produced. Also, the default treatment of forms with omitted material uses +r3, rather than the usual +r1. Only alphanumeric characters are counted and the forms xxx, yyy, and www are excluded, as are forms beginning with & or 0 and any material in comments. The last two analyses are computed from the %mor line. There, the forms with xxx, yyy, or www are also excluded. For the segments of WDLEN that run off the %mor line, you should use the form of the +/-s switch that conforms with the syntax of the %mor line. This form uses +sm, rather than something like +s"n|*". You can best understand the syntax of +sm by typing:

```
vocd +sm
```

The WDLEN command allows 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 only option unique to WDLEN is +d that allows you to output the results in a format that can be opened directly from Excel. Information regarding the additional options shared across commands can be found in the chapter on Options.

8 Profiling Commands

This section describes CLAN's profiling commands. They include

1. **C-NNLA**: Northwestern Narrative Language Analysis
2. **C-QPA:** Quantitative Production Analysis
3. **CORELEX**: Core lexicon analysis for 5 AphasiaBank tasks
4. **DSS**: Developmental Sentence Score
5. **EVAL**: computation of a wide range of indices for aphasia
6. **FluCalc**: computation of a wide range of indices for stuttering
7. **IPSyn**: Index of Productive Syntax
8. **KIDEVAL**: computation of a wide range of indices for child language
9. **MORTABLE**: computation of occurrences of grammatical morphemes
10. **SUGAR**: Sampling Utterances and Grammatical Analysis Revised

8.1 C-NNLA

This command provides an automatic computation of the Northwestern Narrative Language Analysis profile (Thompson et al., 1995). C-NNLA has been designed to compute measures in accordance with the rules provided in the NNLA manual. Although a few individual lexical codes are different from those given in the NNLA manual, all automatically computed measures have been shown to have an acccuracy comparable to that achieved manually by highly trained and experienced NNLA coders. Currently the program is implemented for English only. The command depends on the presence and accuracy of %mor and %gra lines, as described in the MOR manual.

8.1.1 CHAT File Format Requirements

For accurate computation of outcome measures according to NNLA rules, the following CHAT conventions must be followed.

1. **Exclusions**. NNLA rules call for several exclusions in computing outcome measures. Many of these exclusions are automatic. CLAN already excludes repetitions marked with [/], revisions marked with [//], fillers transcribed with &-, and fragments transcribed with &+. The C-NNLA command also automatically excludes the following conjunctions when they are used the beginning of an utterance: *and, but, or, then, so, well, and then, but then, and so*. To exclude other words from C-NNLA analysis, as per the NNLA manual (e.g. interjections, comments), transcribers must manually insert the [e] code after the word(s) to be excluded. For entire utterances to be excluded, use the [+ exc] code after the final punctuation. Here are three examples:

```
*PAR:  the prince says oh [e] it is you.
*PAR:  <I think> [e] her name was Cinderella.
*PAR:  I can't do this.  [+ exc]
```

2. **Utterance level coding**. Two codes are needed to mark grammatically flawed [+ gram] and semantically flawed [+ sem] utterances, as per the NNLA manual.

```
*PAR: she was a really nice dress. [+ sem]
*PAR: he has a wonderful time. [+ sem](talking about Cinderella)
*PAR:  looking at the clock. [+ gram]
*PAR:  it is just stepmother and three stepsisters. [+ gram]
```

3. **Morphological error coding**. The error-coding chapter in the CHAT manual provides word-level error codes that can be used in CHAT files for phonological, semantic, neologistic, and morphological errors. For accurate computation of several morphological outcome measures in C-NNLA, it is important to mark the morphological errors, as in the following examples. Note, that the target word is entered in square brackets with a single colon, followed by the word-level error code.

```
*PAR: both was [: were] [* m:vsg:a] very mean. [+ gram]
*PAR: it felled [: fell] [* m:+ed] out. [+ gram]
*PAR: she was push [: pushing][* m:0ing] pins and needles. [+ gram]
```

8.1.2 C-NNLA Output

All C-NNLA measures are calculated from the %mor and %gra lines. All measures ignore unintelligible utterances and utterances with the [+ exc] postcode. The basic format of the command is: **c-nnla +t*PAR filename.cha**

The command will generate one Excel spreadsheet with all outcome measures. (Note: the .xls file is actually a CSV text file, so it is advisable to save these as .xlsx files in Excel.). Here is a list of the measures computed in C-NNLA.

1. duration: total time of the sample in seconds. Note: If the transcript is not linked, this will not be calculated. An alternative is to add a TIME DURATION line to the ID lines in the transcript (see the TIMEDUR section in this manual).
2. words per minute: total words divided by total time for speaker
3. total utterances: # of utterances
4. total words: # of words (tokens)
5. MLU words: MLU in words (excludes utterances with any unintelligible content transcribed as xxx)
6. open-class: # of open-class words -- all nouns, all verbs excluding auxiliaries and modals, all adjectives, all adverbs
7. % open-class/all words
8. closed-class words: # of closed-class words -- all other words besides open-class words, with the exception of words coded as communicators (e.g., “yeah”) and onomatopoeia (e.g., “woofwoof”)
9. % closed-class words/all words
10. nouns: # of nouns
11. % nouns/all words
12. verbs: # of verbs, including copulas and participles
13. % verbs/all words
14. noun/verb: ratio of nouns to verbs
15. adj: # of adjectives
16. adv: # of adverbs
17. det: # of determiners (det:art and det:dem)
18. pro: # of pronouns (excluding "wh" interrogative pronouns and "wh" relative pronouns) and possessive determiners
19. aux: # of auxiliaries
20. conj: # of conjunctions (excluding "wh" conjunctions) and coordinators
21. complementizers: # of complementizers

22. modals: # of modals and modal auxiliaries
23. prep: # of prepositions
24. negation markers: # of negatives and "no" communicators
25. infinitival markers: # of infinitives
26. quantifiers: # of quantifiers, numbers (det:num), and post (e.g., "all", "both")
27. wh-words: # of "wh" relative pronouns, "wh" interrogative pronouns, "wh" conjunctions, and interrogative determiners
28. comparative suffixes: # of words with -CP
29. superlative suffixes: # of words with -SP
30. possessive markers: # of words with -POSS
31. regular plural markers: # of words with -PL
32. irregular plural forms: # of words with &PL
33. 3rd person present tense markers: # of words with -3S
34. regular past tense markers: # of words with -PAST
35. irregular past tense markers: # of words with &PAST
36. regular perfect aspect markers: # of words with -PASTP
37. irregular perfect participles: # of words with &PASTP
38. progressive aspect markers: # of words with -PRESP excluding gerunds (nouns)
39. % correct regular inflection: numerator = of all regular inflected verbs, copulas, participles, auxiliaries, and "does" modal that do not have any morphological error codes next to them; denominator = all regular inflected verbs, copulas, participles, auxiliaries, and "does" modal
40. % correct irregular inflection: numerator = numerator = of all irregular inflected verbs, copulas (except for "is"), participles, auxiliaries (except for "is" and "are"), and "did" modal that do not have any morphological error codes next to them; denominator = all irregular inflected verbs, copulas (except for "is"), participles, auxiliaries (except for "is" and "are"), and "did" modal
41. % sentences produced: numerator = utterances that have at least one verb, copula, modal, or participle; denominator = all utterances counted on speaker tier except non-word utterances
42. % sentences with correct syntax, semantics: numerator = utterances that have at least one verb, copula, modal, or participle and do not have a [+ gram] or [+ sem] post-code; denominator = all sentences (numerator from % sentences produced)
43. % sentences with flawed syntax: numerator = utterances that have a [+ gram] post-code and have at least one verb, copula, modal, or participle; denominator = all sentences (numerator from % sentences produced)
44. % sentences with flawed semantics: numerator = utterances that have a [+ sem] post-code and have at least one verb, copula, modal, or participle; denominator = all sentences (numerator from % sentences produced)
45. sentence complexity ratio: numerator = # of sentences that have at least 1 verb, copula, modal, or participle and have at least one CSUBJ, COMP, CPRED, CPOBJ, COBJ, XJCT, CJCT, CMOD, XMOD on the %gra tier; denominator = sentences that have at least one verb, copula, modal, or participle and do not have any of those codes on the %gra tier (Note: there are some additional rules about this computation that can be found at the C-NNLA link at the AphasiaBank website in the Discourse Analysis section)

46. # embedded clauses/sentence: numerator = # sentences with one embedded clause + # of sentences with two embedded clauses x 2 + # of sentences with three embedded clauses x 3, etc.; denominator = # of sentences with zero embeded clauses + # of sentences with one embedded clause, two embedded clauses + # of sentences with three embedded clauses, etc. For details on how embedded clauses are counted on the %gra tier, see the C-NNLA link at the AphasiaBank website in the Discourse Analysis section.

8.2 C-QPA

This command provides an automatic computation of the Quantitative Production Analysis profile (Berndt, Wayland, Rochon, Saffran, & Schwartz, 2000). C-QPA has been designed to compute measures in accordance with the rules set out by the QPA authors. Currently the program is implemented for English only. Use of the command requires the presence of accurate %mor and %gra lines.

8.2.1 CHAT File Format Requirements

For accurate computation of outcome measures according to QPA rules, the following CHAT conventions must be followed.

1. **Sentence exclusions.** Use the [+ exc] post-code after final punctuation to exclude:
    a. sentences that respond to directive questioning from the Examiner
    b. comments made by PAR (e.g., " I can't remember what happens next.")
    c. sentences that consist only of direct discourse markers (e.g., "he said", "the fairy godmother told her")

2. **Required determiners.** Mark any missing **required** determiners for nouns with 0det, for example:

```
*PAR:  she tried to fit 0det shoe on her foot.
*PAR:  0det sisters were mean to Cinderella.
*PAR:  mother is washing dishes at 0det sink.
```

3. **Required subjects.** Mark any missing subjects in sentences with a [+ 0subj] post-code. Do not use this code for imperatives like "clean the floor". Example:

```
*PAR:  make her his wife. [+ 0subj]
```

4. **Imperatives.** Mark any imperatives preceded by proper nouns or pronouns with vocative or summons marker ‡ (F2-v), for example:

*PAR: Cinderella ‡ clean the floor.

*PAR: hey ‡ watch out!

5. **Syntax errors**. Mark any sentences NOT syntactically well-formed with [+ gram] post-code. Semantically anomolous sentences are okay. Examples:
    a. missing obligatory arguments
    b. agreement errors
    c. deleted elements

6. **Lexical exclusions.** Use [e] next to an individual word or next to a phrase in angle brackets, for direct discourse markers, habitually used starters, and anything else not automatically excluded by CLAN. Examples:

```
*PAR:  her fairy godmother appeared <and said> [e].
*PAR:  okay [e] his stool is tipping over.
```

By default, CLAN automatically excludes repetitions, revisions, fillers, and sound fragments from analysis. In adherence to the QPA rules, C-QPA will also automatically exclude the following items:

1. frozen words elements -- once upon a time, happily ever after
2. words associated with these error codes -- [* n:uk], [* s:uk], [* n:uk:s]
3. habitually used starters -- and, but, so, then, and then, so, well
4. sentences with with the [+ exc] post-code, words followed by the [e] code

8.2.2 C-QPA Output

All measures are calculated from the %mor and %gra lines. All measures ignore unintelligible utterances and utterances with [+ exc] postcode. The basic format of the command is: **c-qpa +t*PAR filename.cha**

The command will generate two .xls spreadsheets: the sentence-by-sentence analysis spreadsheet, and the summary spreadsheet. (Note: the .xls files are text files, so it is advisable to save them as .xlsx files in Excel.) Here is a list of the measures computed in C-QPA.

Analysis Spreadsheet

1. utterance – actual speaker utterance
2. sentence utterance (1,0) – 1 for sentences that have a noun (SUBJ) and main verb (ROOT); 1 for imperative sentences (ROOT without [+0 subj] post-code); 0 for sentences that do not have a noun and main verb
3. other utterance (1,0) – 1 for sentences that do not have a noun (SUBJ) and main verb (ROOT); 0 for sentences that do have a noun and main verb
4. # narrative wds – total words, tokens
5. # open class wds -- all nouns, verbs (excluding auxiliaries and modals), participles, adjectives, and adverbs with -ly endings
6. # nouns – all nouns
7. # Ns req det (NRDs) -- # of nouns, excluding proper nouns, nouns with numbers (e.g., "three girls"), 2nd noun in compound noun phrase ("queen" in "the king and queen")
8. # NRDs w/dets -- # of nouns with determiners (articles, demonstratives, possessives) minus nouns preceded by 0det
9. # pronouns -- # of pronouns (excluding relative pronouns, interrogative pronouns, demonstrative pronouns, existential pronouns) and possessive determiners
10. # verbs -- # of verbs, copulas, and participles

NOTE: All measures below (11-19) are not computed if Sentence Utterance = 0

11. # matrix verbs -- # of main verbs (verb, copula, or participle) in the sentence, which

will only be more than 1 for conjoined verb phrases
12. AUX score -- # of copulas plus # of verbs (except if it immediately follows infinitive "to") plus # of auxiliaries plus # of modals plus # of participles plus # of inflections (except for copula and auxiliary "is") plus # of "not" immediately following copula, modal, or auxiliary
13. # embeddings -- # of these codes on %gra tier: CSUBJ; CJCT; COBJ; CMOD; COMP if SUBJ directly ties to it; CPRED if SUBJ directly ties to it; XJCT if SUBJ directly ties to it; XMOD if SUBJ directly ties to it (%gra codes are explained in the MOR manual, section 10.6 MEGRASP)
14. S well-formed -- # of utterances that do not have [+ gram] post-code
15. # phrases SNP -- # of nouns and pronouns on the %mor tier that correspond to these grammatical relations on %gra tier: SUBJ directly linked to ROOT; PRED directly linked to ROOT; COORD directly linked to CONJ linked to SUBJ linked to ROOT; POBJ directly linked to NJCT linked to SUBJ
16. # phrases VP -- # of the following: ROOT; ROOT with CONJ followed by COORD where CONJ links to ROOT and COORD links to CONJ; ENUM linked to ROOT; LINK that corresponds to coordinator on the %mor tier and links to CJCT which links to ROOT on the %gra tier
17. # open class words + pron SNP -- # of nouns, copulas, adjectives, pronouns (indefinite, personal, subject, demonstrative), numbers as determiners, adverbs with -ly endings in the SNPs; count 1 for imperatives (where ROOT is the first or only code on the %gra tier or follows a BEGP, with no 0subj post-code)
18. # open class words + pron VP -- # of the following parts of speech that correspond to ROOT or are linked to ROOT on the %gra tier: nouns, verbs, participles, copulas, adjectives, pronouns (indefinite, subject, object, demonstrative), and adverbs with -ly endings
19. # wds in Ss -- # of words in each utterance (where Sentence Utterance = 1)

Summary Spreadsheet

(Note: Several of these measures are the same as the ones in the Analysis Spreadsheet above and will not be re-defined here.)

1. duration -- total time of the sample in seconds. Note: If the transcript is not linked, this will not be calculated. An alternative is to add a TIME DURATION line to the ID lines in the transcript (see the TIMEDUR section in the CHAT manual).
2. # narrative words
3. # words per minute -- total narrative words divided by total time for speaker
4. # open class words
5. # closed class words -- all other parts-of-speech that are not open class, excluding onomatopoeia and communicators
6. proportion closed class words -- # closed class words divided by # narrative words
7. nouns
8. # NRDs
9. # NRDs w/determiners
10. DET index -- # NRDs with determiners divided by # NRDs
11. # pronouns
12. proportion pronouns -- # pronouns divided by # pronouns plus # nouns
13. # verbs

14. proportion verbs -- # of verbs divided by # of pronouns plus # of nouns
15. # matrix verbs
16. total aux score
17. aux complexity -- [total aux score divided by # of matrix verbs] minus 1
18. # Ss -- # of the following: utterances that include a SUBJ linked to ROOT on the %gra tier and is not an incomplete utterance (trailed off, interrupted); utterances with ROOT as the first or only code on the %gra tier or the first code following BEGP (with no 0subj post-code)
19. # words in Ss
20. proportion words in Ss -- # of words in sentences divided by # of narrative words
21. # well-formed Ss
22. proportion well-formed Ss -- # of well-formed sentences divided by # of sentences
23. # SNPs
24. # words in SNPs
25. mean SNP length -- # of words in SNPs divided by # of SNPs
26. SNP elaboration -- [# of words in SNPs divided by # of SNPs] minus 1
27. # VPs
28. # words in VPs
29. mean VP length -- # of words in VPs divided by # of VPs
30. VP elaboration -- [# of words in VPs divided by # of VPs] minus 1
31. S elaboration -- SNP elaboration plus VP elaboration
32. # embeddings
33. embedding index -- # of embeddings divided by # of sentences
34. # utterances
35. mean utterance length -- # of narrative words divided by # of utterances

8.3 CORELEX

This command provides automatic computation of core lexicon lists for the five AphasiaBank Discourse Protocol tasks, as published in Dalton et al. (2020). The command will automatically extract the appropriate gem (task) and create a spreadsheet showing which words from the core lexicon were used. The "Types" column in the spreadsheet will show how many words from the list were used. The other columns will show which specific words from the list were used and how frequently. Be sure to save the spreadsheet as an .xlsx Workbook in Excel.

8.3.1 CHAT File Format Requirements

For accurate computation, the following CHAT conventions must be followed:

1. The transcript must have a gem heading (e.g., @G:<tab>Window) at the beginning of the task. The current list of gem headings for the CORELEX command is: Window, Umbrella, Cat, Cinderella, Sandwich.
2. Enter the command — **corelex +lcat +t*par filename.cha** . Substitute the appropriate task name (e.g., sandwich, window) for "cat" in the above example. Use *.cha to analyze all files in a folder.
3. The command counts words in utterances marked with [+ exc] codes (which is how the norms for Corelex were computed), but if you used that code in your transcripts

and want those utterances excluded, use this command -- **corelex +lcat +t*par -s"<+ exc>" filename.cha** .

If you want to compare your results to the norms reported in Dalton et al. (2020), you need to do two steps before running the CORELEX command because the norms included revised words and excluded target replacements for semantic paraphasias. The CORELEX program counts lemmas on the %mor tier (to capture different forms of a word — e.g., "was" for "be"), and the %mor tier excludes revisions and includes target replacements. To fix that (include revised words, exclude target replacements):

1. Run this command on your CHAT file(s) -- **chstring +q1 filename.cha** -- to remove revision codes in the transcript ([//]) and replace target replacements for semantic paraphasias (e.g., grandmother [: godmother]) with double colons instead of single colons. That will create a new file with a .chstr.cex extension.
2. Re-run the MOR command -- **mor filename.chstr.cex**-- on the new file(s).
3. Run the CORELEX command -- **corelex +lcat +t*par filename.chstr.cex** -- on the new file(s). Now, CORELEX will count every word the participant said and will not count any target replacements that may be in the transcripts for semantic paraphasias. Remember to substitute the appropriate task name (e.g., sandwich, window) for "cat" in the above example and use *.cex (or *.chstr.cex) instead of filename.chstr.cex for multiple files in a folder.

Note: If you are transcribing from scratch and your transcript does not have revision markings or target replacements, you would not need to do this to be able to compare your results with the norms.

8.4 DSS

This program provides an automatic computation of the Developmental Sentence Score (DSS) (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.

8.4.1 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 to be included.
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 [+ dsse] postcode must be included for these utterances. This exclusion will also apply to

KIDEVAL.

8.4.2 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 the xxx, yyy and www 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, any sentences containing a [+ dsse] post-code are excluded.
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.

8.4.3 Automatic Calculation of DSS

To compute DSS, the user must first complete a morphological analysis of the file using the MOR program. The command *mor *.cha* runs the MOR, PREPOST, POST, POSTMORTEM, and MEGRASP commands one after another to create a %mor line for part-of-speech analysis and a %gra line for grammatical relation analysis. To make sure

that MOR recognizes each word properly, it is best to first run the *mor +xb *.cha* command. Further details about running MOR can be found in the third TalkBank manual on morphosyntactic analysis. Once the disambiguated %mor is created, the user can run DSS to compute the Developmental Sentence Analysis. A basic DSS command has this shape:

```
dss +b*CHI +le *.cha
```

8.4.4 Sentence Points

DSS assigns a sentence point to each sentence if it "meets all the adult standard rules" (Lee, p. 137). Sentence points are withheld for all errors and omissions. Errors, including neologisms, should be marked with the [*] codes used in the CHAT error coding system. Omissions should be marked with a code such as 0det or 0aux, for example, to mark a missing determiner or a missing auxiliary.

Warning: MOR cannot distinguish between non-complementing infinitive structures, such as *I stopped to play*, which receives 3 points as secondary verb and infinitival complement structures, such as *I had to go*, which receive 5 points as secondary verbs. When the latter occurs, you must edit the output table to assign two more points.

8.4.5 DSS Output

Once all 50 sentences have been assigned sentence points, the DSS program automatically generates a table. Each sentence is displayed on the leftmost column of the table with the corresponding point values, sentence point, and the DSS score. 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: 3.4   (i.e. 17/5)
```

8.4.6 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
3a no, some, more, all, lot(s), one(s), two (etc.), other(s), another
3b something, somebody, someone
4 nothing, nobody, none, no one
7a any, anything, anybody, anyone
7b every, everyone, everything, everybody
7c 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)
3a plurals: we, us, our(s) they, them, their
3b 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)

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

Secondary Verbs (SV) (D)

2a five early developing infinitives
2b I wanna see, I'm gonna see, I gotta see, Lemme see, Let's play
3 noncomplementing infinitives: I stopped *to play*
4 participle, present or past: I see a boy *running.* I found the vase *broken*.
5a early infinitives with differing subjects in basic sentences:

I want you *to come*
5b later infinitival complements: I had *to go*
5c obligatory deletions: Make it [*to*] go
5d infinitive with wh-word: I know what *to get*
7 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.
4 can't don't
5 isn't won't
7a uncontracted negatives with have: I have not eaten it.
7b any other pro-aux + neg forms: *you're not, he's not*
7c any other aux-negative contractions: *aren't, couldn't*

Conjunction (CNJ) (F)

3 and
5a but
5b so, and so, so that
5c or, if
8a where, when, how, while, whether, (or not), till, until, unless, since,
before, after, for, as, as + adjective + as, as if, like, that, than
8d wh-words + infinitive: I know *how* to do it.

Interrogative Reversal (IR) (G)

1 reversal of copula: *is*n't *it* red?
4 reversal of auxiliary be: *Is* he coming?
6a obligatory -do, -does, -did *Do* they run?
6b reversal of modal: *Can* you play?
6c tag questions: It's fun *isn't* it?
8a reversal of auxiliary have: *Has he* seen you?
8b reversal with two auxiliaries: *Has he been* eating?
8c reversal with three auxiliaries: *Could he have been going?*

Wh-question (WHQ) (H)

2a who, what, what + noun
2b where, how many, how much, what....do, what....for
4 when, how, how + adjective
7 why, what it, how come, how about + gerund
8 whose, which, which + noun

8.4.7 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., 2013) 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 +djpn.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 JMOR06 and WAKACHI2002 v.5.

Verb Final Inflection (Vlast)

1 PAST (*tabeta*), PRES (*taberu*), IMP:te (*tabete!*)
2 HORT (*tabeyoo*), CONN (*tabete...*)
3 COND:tara (*tabetara*)
4 CONN&wa (*tabecha*), GER (*tabe*), NEG&IMP:de (*tabenaide!*)
5 IMP (*tabero*), NEG&OBL (*tabenakucha*)

Verb Middle Inflection (Vmid)

1 COMPL (tabechau), NEG (tabenai), ASP/sub|i (tabeteru/tabete iru)
2 DESID (tabetai), POT (tabererareru/taberereru), POL (tabemasu), sub|ku (*tabete kuru*), sub|ik (*tabete iku*)
3 sub|mi (*tabete miru*), sub|ar (*tabete aru*), sub|ok (*tabete oku*), sub|age (*tabete ageru*)
4 PASS (tabererareru)
5 sub|moraw (*tabete morau*), sub|kure (*tabete kureru*)

Adjective Inflection (ADJ)

1 A-PRES (*oishii*)
3 A-NEG- (*oishikunai*), A-ADV (*oishiku*)
4 A-PAST (*oishikatta*)

Copula (COP)

1 da&PRES (*da*)
3 de&wa-NEG-PRES (*janai*), de&CONN (*gakusee de*)
4 da-PAST (*datta*), da&PRES:na (*gakusee na no*), ni&ADV (*gakusee ni naru*)
5 de&CONN&wa (*kami ja dame*)

Adjectival Nouns + Copula (AN+COP)

4 AN+da&PRES (*kiree da*), AN+ni&ADV (*kiree ni*), AN+da&PRES:na (*kiree na*)

Conjunctive particles (CONJ ptl)

2 kara=causal (*kiree da kara*)
3 to (taberu to ii), kara=temporal (kaette kara taberu), kedo (tabeta kedo)
4 shi (taberu shi), noni (tabeta noni)

Conjunctions (CONJ)

4 datte (datte tabeta mon), ja (ja taberu), de/sorede (de tabeta), dakara (dakara tabeta)
5 demo (demo tabechatta)

Elaborated Noun Phrases (NP)

2 N+no+(N) (ringo no e), A+N (oishii ringo)
3 N+to+N (ringo to nashi), Adn+N (ironna ringo), V+N (tabeta ringo)

5 AN+na+N (kiree na ringo), V+SNR (tabeta no ga chigau)

Formal Nouns (FML)

4 koto (tabeta koto aru)
5 hoo (tabeta hoo ga ii)

Compounds (COMP)

1 N+N (geta+bako)
5 PROP+N (Nagoya+eki), V+V (tabe+owaru), N+V (jagaimo+hori)

Case and Post Particles (CASE)

1 ga (ringo ga oishii), ni (ringo ni tsukeru)
2 to (Papa to asonda), de (ie de taberu)
3 o (ringo o taberu), kara (ringo kara deta)
5 made (ie made hashiru)

Topic, Focus, Quotation Particles (TOP, FOC, QUOT)

1 wa (ringo wa ii), mo (ringo mo ii)
2 tte (ringo tte nani?)
3 dake (ringo dake), to (quot) (ringo to iu to…)
5 kurai (sannin kurai ita), shika (sannin shika inai)

Adverbs (ADV)

2 motto (motto taberu), moo (moo tabenai), mata (mata taberu)
3 mada (mada tabetenai), chotto (chotto taberu), ippai (ippai taberu)
4 ichiban (ichiban oishii), nanka (nanka oishii), sugu (sugu taberu)
5 yappari (yappari taberu), sakki (sakki taberu)

Sentence Final Particles (SFP)

1 yo (taberu yo), no (taberu no), ne (taberu ne)
2 kanaa (taberu kanaa), mon (taberu mon), ka (taberu ka), naa (taberu naa)
3 no+yo (taberu no yo)
4 yo+ne (taberu yo ne), kke (tabeta kke)

Sentence Modalizers (SMOD)

3 desu (oishii desu)
4 mitai (ringo mitai), deshoo (tabeta deshoo)
5 yoo (ikeru yoo ni suru), jan (tabeta jan)

Miyata, S., MacWhinney, B. Otomo, K. Sirai, H., Oshima-Takane, Y., Hirakawa, M., Shirai, Y., Sugiura, M. and Itoh, K. (2013). Developmental Sentence Scoring for Japanese. First Language 33, 2, x-x. http://fla.sagepub.com/content/early/recent www2.aasa.ac.jp/people/smiyata/papers/Miyata&al_DSSJ_FirstLanguage2013.pdf

8.4.8 How DSS works

DSS relies on a series of rules stated in the rules file in CLAN/lib/dss, such as eng.cut. These rules are listed in a "STARTRULES" line at the beginning of the file. They are fired in order from left to right. Each rule matches one of the top-level categories in the DSS. For example, the D rules (fourth letter of the alphabet) match the SV category that is the

fourth category in the DSS. Within each rule there is a series of conditions. These conditions each 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. This means that the asterisk is a wild card for "anything"; the plus means "or", and the up arrow means "followed by". DSS goes through the sentence one word at a time. For each word, it checks for a match across all 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. Specifically, the C1 condition for main verbs is ordered after C2 and C7 for this reason. 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.

8.4.9 Unique Options

DSS has the following unique options:

+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 criterion is used in the development of the corpora. Any sentences containing xxx yyy and www codes will be excluded from the corpora.

+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.

8.5 EVAL and EVAL-D

EVAL is a CLAN program that analyzes transcripts and yields a spreadsheet displaying many language characteristics. EVAL is used with data collected with the AphasiaBan

protocol and EVAL-D is a variant of EVAL used for data collected with the DementiaBank protocol The program can be used in three ways:

1. Both EVAL and EVAL-D can analyze a participant's performance on a discourse task.
2. EVAL can analyze a participant's performance of a discourse task from the AphasiaBank protocol and compare the results to those of a reference group from that database. EVAL-D can do the same for DementiaBank protocol data. The resulting spreadsheet displays the participant's analysis side-by-side with the mean scores of the comparison group and indicates where the participant and the comparison group differ by one or more standard deviations.
3. The programs can analyze the baseline performance, then you can re-administer and analyze the discourse task after a period of therapy. The spreadsheet displays the pre- and post- therapy results side-by-side, allowing a comparison of performance at different time points.

The use of EVAL is described in tutorial screencasts available from http://talkbank.org/screencasts .

8.5.1 Explanation of EVAL Measures

All measures are calculated from the %mor tier except for retracing, repetition, and duration which are calculated from the speaker tier. FREQ numbers are calculated based on lemmas.

1. Duration: total time of the sample in hours:minutes:seconds. For simplified EVAL transcripts, which are not linked with time bullets, users who want this information need to add a TIME DURATION line to the ID lines in the transcript (see EVAL Manual, Section 4.c). For linked transcripts EVAL calculates this item automatically.
2. Total Utts: total utterances. Includes all utterances used in computing MLU, plus utterances with xxx (unintelligible). Excludes non-word utterances, for example a gesture only, coded as *PAR: &=head:yes.
3. MLU Utts: number of utterances used to compute MLU. Excludes utterances with xxx code and non-word utterances (*PAR: 0). Adding +s to the EVAL command counts utterances and words in utterances with xxx.
4. MLU Words: MLU in words. Excludes words in utterances with xxx, yyy or www codes.
5. MLU Morphemes: MLU in morphemes. Excludes morphemes in utterances with xxx, yyy or www codes.
6. FREQ types: total word types as counted by FREQ. The default does not include repetitions and revisions.
7. FREQ tokens: total word tokens as counted by FREQ. The default does not include repetitions and revisions.
8. FREQ TTR: type/token ratio
9. Words/min: words per minute (FREQ tokens/Duration converted to minutes)
10. Verbs/Utt: verbs per utterance: (roughly corresponds to clauses per utterance). Includes verbs, copulas, and past or present participles (that appear with auxiliaries); does not

include modals.

11. % Word Errors: percentage of words that are coded as errors [*]. These numbers are based on the speaker tier and include revised and repeated material. This item can be displayed as a raw number rather than a percentage by adding +o4 to the EVAL command.
12. Utt Errors: number of utterances coded as errors. For EVAL transcriptions the code is [+ *]. For transcripts with full error coding, the [+ *] code subsumes utterance-level errors from the Error Coding section of the CHAT manual ([+ jar], [+ es], [+ cir], [+ per], and [+ gram]). This code is intended to flag errors for closer examination. It does not allow accurate comparison with the control database, since any utterance-level errors of controls are not coded.
13. Density: measure of propositional idea density. This measure was adapted with permission, from CPIDR3 (Computerized Propositional Idea Density Rater, third major version), developed under the direction of Michael A. Covington, University of Georgia Artificial Intelligence Center, August 17, 2007. CPIDR replicates Turner and Green's rules (1977) for extracting propositions from text, based on the theory of Kintsch (1974) and the work of Snowdon et al. (1996) who who showed that propositional idea density can be approximated by the number of verbs, adjectives, adverbs, prepositions, and conjunctions divided by the total number of words. For a list of the words counted as propositions from each CHAT file, add +e2 to the EVAL command.

The next 15 items (#14-28) are expressed as percentages by default. For parts of speech (e.g., nouns, verbs, auxiliaries, modals, adverbs, adjectives), the denominator is total words; for % plurals, the denominator is total nouns; for items 19-23, the denominator is total verbs. To display these results as raw numbers instead of percentages, add +o4 to the EVAL command.

14. % Nouns
15. % Plurals
16. % Verbs: includes those tagged by MOR as verb, participle, and copula
17. % Aux: auxiliaries
18. % Mod: modals
19. % 3S: third person singular
20. % 1S/3S: identical forms for first and third person (e.g., I was, he was)
21. % Past
22. % PastP: past participle
23. % PresP: present participle
24. % prep: prepositions
25. %adv: adverbs
26. %adj: adjectives
27. % conj: conjunctions
28. % det: determiners (includes articles, demonstratives, interrogatives, numbers, and

posessives used as determiners)

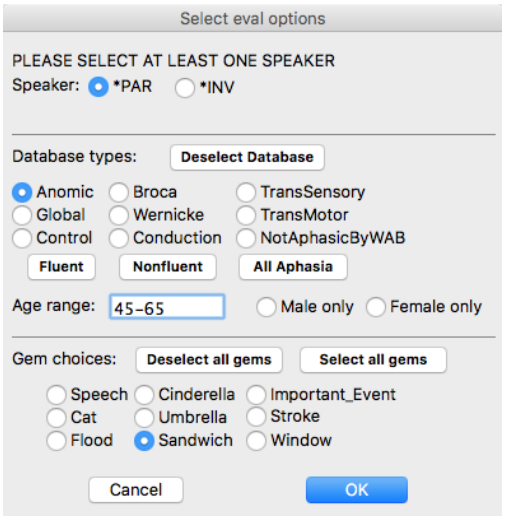
29. % pro: pronouns
30. noun/verb ratio: total # of nouns ÷ total # of verbs (excluding auxiliaries and modals)
31. open/closed ratio: total # of open class words ÷ total # of closed class words; open class includes all nouns, adverbs, adjectives, and all verbs including copulas and participles but excluding auxiliaries and modals; closed class includes all other parts of speech; co(municator) and on(omatopoeia) are excluded from both counts.
32. #open-class: total # of open class words
33. #closed-class: total # of closed class words
34. retracing [//]: number retracings (self-corrections or changes)
35. repetition [/]: number of repetitions

By default, sentences marked by [+ exc] are excluded from the analysis. For cases where target replacements are in the transcript next to error productions with missing morphemes (e.g., he is kick [: kicking] [* m:0ing] the ball), the EVAL and MORTABLE programs will reflect the speaker's morphological production (e.g., v|kick) and not count anything that was not produced (e.g., part|kick-PRESP). For cases where target replacements are in the transcript next to error productions for superfluous morphemes (e.g., there is one birds [: bird] [* m:+s] in the tree), the EVAL and MORTABLE programs will not count the superfluous morphemes (e.g., n|bird-PL) because they were not used correctly.

8.5.2 EVAL Demo

To run a demo of the EVAL program, follow these steps:

1. To start the comparison, push the **Progs** (for programs) button in the CLAN commands window, and select **Eval** from the drop-down menu of CLAN commands. Push the **Option** button, and you will see a list of the files in your working directory folder. Navigate to the /examples/eval folder, and double click on **eval_demo.cha** and it will be displayed on the rightside of the box. When you have chosen the right file for analysis, click on **Done**.
2. A new window, shown below, will appear for selecting EVAL options. For this demonstration, you want to compare eval_demo.cha to a sample from the AphasiaBank database. Since eval_demo.cha is a transcript of an anomic aphasic person, we suggest choosing Anomic as the aphasia type for comparison. Under Age Range, since the eval_demo participant is 55 years old, we suggest choosing 45-65 (in this format). If you want only males or only females, choose the appropriate circle. If you want both genders included, leave those circles blank, which is the choice we suggest here. Next select the AphasiaBank discourse task you want to compare to eval_demo.cha. The eval_demo.cha file is a transcript of the Sandwich Gem, so select Sandwich under Gem choices. Your window will now look like this:



3. Press the OK button, and the Commands window will echo this full command you have constructed:

   eval @ +t*PAR: +d"Anomic45-65" +g"Sandwich" +u

4. Press Run and CLAN will produce an output file called eval_demo.xls, as listed in the CLAN Output window. Click three times on that last line and the results will open in Excel. If Excel warns you about opening the files, just say "yes". The columns list various outputs in terms of indices and part of speech frequencies.

8.5.3 EVAL Output

After triple-clicking on the output from this demo, you open up a spreadsheet. Below is the first part of that spreadsheet. Data from the eval_demo transcript are displayed across the first row, and the information for the comparison database is displayed in the rows below. The top line of the first column identifies the name of the file being analyzed. Below that is a row specifying the percentage of a standard deviation that eval_demo differs from the database. When the difference is a standard deviation or more, the next row indicates this with one asterisk for one standard deviation or more, and two asterisks for two or more. The next three rows provide information about the comparison database: the mean, the minimum, the maximum, and the standard deviation for each of the language measures EVAL produces. Below this the characteristics of the comparison database are listed (in this case the Sandwich Gem and Anomic participants from 45-65 years old). The next line specifies the number of files in the AphasiaBank database that met the criteria for the comparison database; in this case it was 25. The final line is the CLAN command that was used to generate this spreadsheet. The second column is the ID information for the person whose language is in the transcript.

A	B	C	D	E	F	G
File/DB	Speaker ID	Duration	Total Utts	MLU Utts	MLU Words	MLU Morphe
eval_demo.c	eng|EVALde	0:00:00	5	5	7.6	7.8
+/-SD		-1.672	-0.929	-0.929	1.007	0.883
		*			*	
Mean Database		0:00:59	9.48	9.48	5.418	5.763
Min Database		0:00:10	3	3	2.286	2.286
Max Database		0:02:16	18	18	11	11.833
SD Database		35.474	4.823	4.823	2.166	2.308
+/- SD * = 1 SD, ** = 2 SD						
Database gems: Sandwich						
Database keywords: Anomic|45-65						
# files in database: 25						

8.5.4 Comparing Multiple Transcripts

EVAL can also be used to analyze and compare files (e.g., measure change over time in therapy) without any reference to the AphasiaBank database or the AphasiaBank discourse tasks. As an example, you can use the sample files eval_demo02a.cha and eval_demo02b.cha that are in your examples/eval folder.

Next, push the PROGS button in the CLAN Commands window, and select eval from the dropdown menu of CLAN commands. Push the option button, and you will see a list of the files in your working directory folder. Double click on the files you want to analyze and they will be displayed on the right side of the box. When you have finished your selections, click the Done button.

A new window, shown below, will appear for selecting database options. Under the heading Database types, choose Deselect Database. If your transcriptions contain only one gem or none of the AphasiaBank protocol gems, you can bypass the rest of the choices in the box, and just press OK. If your transcriptions contain more than one gem from the AphasiaBank protocol, and you want to choose one for analysis, choose the circle next to the one you want to analyze and then push OK. The command box will show the appropriate command for your selections. Press run, and the CLAN output will appear on your desktop. Triple click on the last line of the CLAN output to open the Excel spreadsheet displaying your data in one spreadsheet.

EVAL has the following unique options:

+bS: add S characters to the morpheme delimiters list (default: -#~+)

-bS: remove S characters from the morphemes list (-b: empty morphemes list)

+cN: create database file (N == 1: eval +c1 +t"*par")

+dS: specify database keyword(s) "S". The choices are: Anomic, Global, Broca, Wernicke, TransSensory, TransMotor, Conduction, NotAphasicByWAB, Control, Fluent, Nonfluent, Fluent, AllAphasia

+e1: create list of database files used for comparisons

+e2: create proposition word list for each CHAT file

+g: gem tier should contain all words specified by +gS

-g: look for gems in database only

+gS: select gems which are labeled by label S

+n: gem is terminated by the next @G (default: automatic detection)

-n: gem is defined by @Bg and @Eg (default: automatic detection)

+o4: output raw values instead of percentage values

8.6 FLUCALC

The FLUCALC program works very much like KIDEVAL. It tracks the frequencies of the various fluency indicators summarized in the section of the CHAT manual on "Disfluency Transcription" such as retraces, blockings, and initial segment repetitions. Like KIDEVAL, it requires the presence of a %mor tier and the selection of a certain speaker as the target. For example, the +t*PAR switch would select utterances from PAR for analysis. Using the tom.cha file in CLAN's examples/fluency folder, the command for a basic analysis would be;

flucalc tom.cha +t*TOM

The output will go into a file called tom.flucalc.xls which you can open in Excel, after accepting the warning message.

FLUCALC will perform a fluency analysis of a language sample, in both raw counts and percentages of intended words. It will also provide a "beta" weighted disfluency value over words, based on the formula proposed by the Illinois Stuttering Project (Yairi & Ambrose, 1999) for computations made on syllable counts.

You must use fluency codes specified in the CHAT manual and the Clinician's Guide to CLAN. You must run MOR for the appropriate language to then run FluCalc. The same speech/language sample can be used for both language sample analysis (KidEval, EVAL) and fluency appraisal via FluCalc. The output is in CSV (comma-separated variables) spreadsheet format. Spreadsheets in this format can be imported directly to Excel. When opening them, Excel will warn you that the spreadsheet is not in Excel XLS format, but you can just ignore that warning and proceed to open and analyze using Excel.

The user can select to have FLUCALC use either words or syllables as the denominator for the computation of percentages (indicated below by %). For indices that look at intended words, the forms on the %mor line are used, because that line excludes repetitions and nonwords. When counting pauses, FLUCALC ignores pauses that are utterance external. This means that pauses at the beginnings and ends of utterances are not counted, because they often represent features of conversational exchange, rather than purer measures of disfluency.

If the sample has been processed for ASR (automatic speech recognition) using the batchalign system at https://github.com/talkbank there will be a new %wor line that can be used when the +a switch is selected to derive more accurate times for word and pause duration. These additional times are then summarized in 9 columns are the end of the output.

The columns in the output specify the following values (beyond those provided by

header information, such as gender, age, corpus, etc.). The final 9 columns (47-55) are only produced if the input has a %wor line with time values.

1. mor Utts: total utterances in the sample
2. mor Words: total intended words, as given on the %mor line
3. mor syllables: total syllables of the words on the %mor line
4. words/minute: words from the %mor line
5. syllables/minute: syllables in the words in the %mor line
6. #Prolongation: raw count of sound prolongations
7. %Prolongation
8. #Broken word
9. %Broken word
10. #Block
11. %Block
12. #PWR: part-word repetition
13. %PWR
14. #PWR-RU: part-word repetition units, these are sometimes called iterations, or the actual number of repetitions in a part-word repetition unit. This column totals all RUs seen in the sample, for use in the weighted disfluency score
15. %PWR-RU
16. #WWR: whole word repetition
17. %WWR
18. #WWR-mono: monosyllabic repetition
19. %WWR-mono
20. #WWR-RU: repetition units; please see PWR above
21. %WWR-RU
22. #WWR-RU-mono: repetition units; please see PWR above
23. %WWR-RU-mono
24. mean RU = (PWR-RU + WWR-RU) / (PWR+WWR)
25. #Phonological fragment: these are best viewed as abandoned word attempts, e.g. *&+fr- tadpole*, where the speaker appears to change word choices; this code was original to CLAN programs.
26. %Phonological fragment
27. #Phrase repetitions
28. %Phrase repetitions
29. #Word revisions
30. %Word revisions
31. #Phrase revisions
32. %Phrase revisions
33. #Pauses: only utterance-internal unfilled pauses are counted
34. %Pauses
35. #Filled pauses
36. %Filled pauses
37. #TD: typical disfluencies; this is the sum of phrase repetitions, word revisions, phrase revisions, pause counts, phonological fragments, and filled pauses.
38. %TD: total typical disfluencies over total words or total syllables.
39. #SLD: stutter-like disfluencies; this the sum of prolongations, broken words, blocks,

PWRs, and monosyllabic WWRs.

40. %SLD: proportion of stutter-like disfluencies over total intended words
41. #Total (SLD+TD): this sums all forms of disfluency, both stutter-like and typical, seen in the sample
42. %Total (SLD+TD)
43. SLD Ratio: SLD/(SLD+TD)
44. Open_class_ratio: open_class_words with disfluency/total disfluencies

    open_class = n, v, part, cop adj, adv (except adv:int which is closed)
45. Closed_class_ratio: closed_class_words with disfluency/total disfluencies

    closed_class = all items not in the open class, except co and on
46. Weighted SLD: This is an adapted version of the SLD formula for distinguishing between typical disfluency and stuttering profiles in young children. It was originated by Yairi & Ambrose (1999) and referenced against a standard sample of 100 syllables. This formula penalizes the severity of the segment repetition profile as well as the presence of prolonged sounds and blocks, which are virtually absent in any sample of typically fluent speech. The formula is:

    ((PWR + mono-WWR) * ((PWR-RU + mono-WWR-RU)/(PWR + mono-WWR))) +

    (2 * (prologations + blocks))
47. IW_Dur -- total inter-word pause duration for target speaker. FLUCALC requires specifying just one target speaker
48. Utt_dur -- total utterances duration for target speaker
49. IW_Dur / Utt_dur
50. Switch_Dur -- total inter-utterance pause duration time when target utterances follow different speakers
51. #_Switch -- # of times consecutive utterances are from different speakers
52. Switch_Dur/#_Switch
53. No_Switch_dur -- total inter-utterance pause when target utterances follow the same target speaker
54. #_No_Switch -- # of times consecutive utterances are from the same speaker
55. No_Switch_Dur/#_No_Switch

8.7 IPSYN

The IPSyn Command computes the Index of Productive Syntax (Altenberg, Roberts, & Scarborough, 2018; Scarborough, 1990). Computation of this index requires the presence of an accurate %mor line. Currently the program is implemented only for English. The full form of the IPSyn requires 100 acceptable utterances. However, Yang et al. (2021) have shown that IPSyn works equally well with samples of 50 utterances. To allow for this, IPSyn now requires 50 utterances by default and uses the new rule set recommend by Yang et al. The basic form of the command is:

```
ipsyn +t*CHI -leng filename.cha
```

If you wish to run the classic version of IPSyn, then you should use the +o switch whcih will use the old rule set and 100 utterances by default. The classic rule set is designed to conform to the IPSyn-R revised version of the scale from 2018 (Altenberg et al., 2018).

Inaccuracies in CLAN's 2019 version that were noted by Roberts et al. (2020) are largely corrected in the current version.

If you wish to change the treatment of a given utterance, you can use the [+ ipe] postcode to exclude it or the [+ ip] to include it. This exclusion code will also apply to IPSyn inside KIDEVAL. IPSyn excludes the whole utterance if it has an [+ ipe] postcode or if the whole utterance is just xxx, yyy, or zzz. IPSYN also excludes repeated utterances. If an utterance has been spoken verbatim by the speaker before, then it is excluded from analyses, unless the [+ ip] postcode is specified on that utterance. To better see what IPSYN does, run this command on the 98.cha file in the /examples/transcripts/ne32 folder.

```
ipsyn +t*CHI +leng 98.cha
```

The output from this command will be 98.ipsyn.cex. Triple-click on that name in CLAN's output window and you will see how IPSYN assigned points for each relevant utterance in that sample. This same run of the IPSYN command will also produce the file 98.ipcore.cex. You can open that file to see which utterances were included in the IPSyn analysis.

8.7.1 Rule Syntax

The computation of IPSyn relies on a series of rules for point assignment that are given in files in the /CLAN/lib/ipsyn folder such as eng.cut for English or zho.cut for Chinese. Each rule searches through the 50 acceptable utterances for a match to the strings given in the INCLUDE line. If there is a match, then one point is assigned for that rule. In order to assign a second point, the search must then use the information on the DIFFERENTSTEMS line to decide how different a second match must be from the first to be counted. Here is an example for the N6 point which searches for strings like *this one goes* or *my turn is next* with a two-word NP before a verb or preposition:

```
RULENAME: N8
if
INCLUDE:      $MOD ^ $N ^ $V
DIFFERENT_STEMS:    >2
```

The notation of >2 should really be >=2, because the program interprets it as greater than or equal to 2. Given an utterance such as *my dog barks and my cat meows,* this rule would match *my dog barks* for the first point and *my cat meows* for the second point. If the utterance were *my dog barks and my dog runs*, then there would not be a second point, because there would only be one new stem. The matching strings can be anywhere in the 100-utterance sample; it is not necessary that they be in the same utterance. In this case, the requirement that there be two or more different stems would be fulfilled, because two of the stems are different, although one is the same. DIFFERENT_STEMS_POS sets further conditions on the different stems. The statement DIFFERENT_STEMS_POS: 1 means that the first word in the sequence of stems must be different. This occurs in the V3 rule, as shown here:

```
RULENAME:                                                    V3
if
INCLUDE:      |prep                        ^                 $NP
EXCLUDE:      (lookit  ^  this)  +  (in  ^  there)+(on  ^  there)
DIFFERENT_STEMS_POS:                                          1
if
```

```
POINT:  2
INCLUDE:      (in ^ there) + (on ^ there)
```

This rule will assign a second point if the two strings are *for the sheep* and *besides the sheep*, but not if the two strings are *for the sheep* and *for a sheep*.

On the INCLUDE line, the ^ or caret indicates that the element after the caret must directly follow the one before. The categories here – $MOD for modifier, $N for noun, and $V for verb – are defined in the eng.cut file.

It is also possible to have additional restrictions placed on the assignment of the points, as in the above example from V3, where the EXCLUDE term disallows a first point for three specific phrases: *lookit this*, *in there*, or *on there*. For the second point, the two exclusions from the first point are now allowed, if neither word matches stems from the first point.

You can see how IPSyn is assigning points by looking at the results for individual words in the *.ipsyn.cex output file. The *.ipcore.cex file shows you which utterances were selected for the analysis.

8.7.2 Cascading Credit

For the classic version of IPSyn, as described in the "credit" column of the Appendix to the IPSyn-R paper (Altenberg et al., 2018), IPSyn-R specifies that a child's successful use of certain higher-order patterns should lead in certain cases to also assigning credit for lower-order patterns. For example, credit on N8 should "cascade" to credit on N4. This means that, if a child places a two-word NP before a verb (N8), then they should also get credit for producing a two-word NP (N4). In addition to these various rules for primary cascading credit, it is also possible to have secondary cascading credit. For example, if a child negates an auxiliary with the form *wasn't*, credit is assigned for Q7. But then this also means that the child can produce a negative between the auxiliary and the verb which requires credit for Q5. However, if Q5 receives credit, then one must also credit Q3, which only requires that the child produce a negation of any type. So, there is a secondary cascade of Q7 > Q5 > Q3. The 11 secondary cascades tracked by CLAN's IPSyn are these: V11 > V9 > V5; V13 > V6 > V5; V16 > V4 > V1; Q6 > Q4 > Q1; Q6 > Q4 > Q2; Q7 > Q5 > Q3; Q11 > Q6 > Q4; Q11 > Q7 > Q5; Q11 > Q8 > Q1; Q11 > Q8 > Q2; S13 > S10 > S5. In addition, there are 11 secondary cascades leading from S6 and S3 back to S1, which is the item requiring a two-word utterance. However, S1 is already so likely to receive full credit, that we do not enforce these further 11 secondary cascades.

8.7.3 IPSYN Rules

This section summarizes the IPSyn rules that govern CLAN's computations of IPSynprofiles. IPSyn includes scores for four dimensions: noun phrases, verb phrases, questions and negation, and sentence structure. Below there are four tables - one for each dimension. For each of these, there are IPSyn rules that make use of higher-level categories that combine basic part-of-speech categories on the %mor line. These categories are articulated in rules that are given in the beginning of the eng-cut file:

$N = |n,|n:prop,|n:pt

$V = |v,|cop,|aux

```
$AMC = |aux,|mod,|cop
$PRO   = |pro:per,|pro:indef,|pro:refl,|pro:obj,|pro:sub
$MOD += |adj,|det:art,|det:poss,|det:num,|det:dem,|qn
$ART = |det:art
$DET = |det:dem
$ADV   = |adv,|adv:loc,|adv:tem,|adv:int
$NP   = $N + $PRO + ($MOD ^ $N) + ($DET ^ $N) + ($ART ^ $N) +
        ($ART ^ $MOD ^ $N) + ($DET ^ $MOD ^ $N)
```

The six columns in these four tables encode the features of each rule. The first column gives the rule number. The second column gives the string match for the rule. Here the plus is used for combination, although in the actual rules the up-arrow symbol is used for this. The third column shows that, if you credit this rule, you should also credit some other rule(s). This is called "cascading credit." The fourth column gives and example structure that matches the rule. The fifth column lists structures to exclude, even if they match the string in the second column. The sixth column gives the criteria that allow for the assignment of a second point for the rule. In this column, the word "stem" refers to the fact that the program uses the lemma or stem from the %mor line, rather than the surface form of the word.

Noun Phrase Points

Level	Include	Also	Example	Exclude	2nd point (min)
N1	$N		dog, Bill		stem different
N2	$PRO		he, mine		stem different
N3	$MOD		red, this		stem different
N4	$MOD + $N		my dog		1 stem different
N5	det ^ $N		the dog, a saw, my orange		1 stem different
N6	$V + $MOD + $N prep + $MOD + n	N4	ride a car, on the road		2 stems different
N7	$N-PL	N1	dog-PL	n:pt	stem different
N8	$MOD + $N + $V	N4	my dog runs		2 stems different
N9	$MOD + $MOD + $N	N4	little red hen		2 stems different
N10	$ADV + $MOD qn +$N	V8	very nice only me	not + X	1 stem different
N11	n-X, adj-X		book-ish, happy-COMP	-POSS, -PL	stem different

Verb Phrase Points

Level	Include	Also	Example	Exclude	2nd point
V1	$V			look, stop	stem different
V2	prep				stem different
V3	prep + $NP	V2	into the truck, at school, of them	lookit this	different part of speech
V4	$NP + v:cop + $NP $NP + v:cop + $MOD	V1	we is monsters he is okay	how are you?	diff form of copula, or diff structure
V5	$CAT +$V		hafta eat		diff aux or v
V6	v:aux + $V	V5	she is coming		diff aux or v
V7	-PROG	V1	running		stem different
V8	adv		now, so, hardly, yet, never	I think so, here, there	stem different
V9	mod + v	V5	can go, will go		stem or head diff
V10	-3S	V1	runs, wants	does, says	stem different
V11	would, could, should, might	V9			stem or head diff
V12	-PAST, -PERF	V1	jumped, fixed		stem diff
V13	v:aux|x-PAST	V6	did say, had gotten		stem or head diff
V14	$ADV + $V $ADV + $MOD	V8			stem different
V151	$N + |cop	V4	I am.		stem different
V152	$N + |aux	V6	he was.		stem different
V153	$N + |mod	V9	Bill could.		stem different
V16	v:cop|x&PAST	V4	were, was, have		stem different
V17	x#v| or –LY		undo, bravely, recut	hardly, really, repeat	stem different or suffix different

Notes:
$CAT = have to, supposed to, going to, want to, got to, better
cop can also be ~cop

Question and Negation Points

Level	Include	Also	Example	Exclude	2nd point
Q1	any question		want some?mine?		stem different
Q2	pro:int		what that? why he ..? why?		stem different
Q3	neg|not or ~neg|not or qn|no		no book, can't		stem or head diff
Q4	pro:int + v	Q1, Q2	who made it?		stem different
Q5	$NP + neg + v	Q3	I not going.		different VP
Q6	pro:wh + aux	Q4	Where was that?	wh as SUBJ	different VP
Q7	aux + neg aux|x~neg|not	Q5	It doesn't work. He might not go.		different VP
Q8	aux + $NP	Q1, Q2	Can I have it? Is she tall?		different Q
Q9	why, when, which, whose	Q1	Why? Which one is taller?		stem different
Q10	Unicode 201E „	Q1, Q2	I'll be the mom„ okay?		tag different
Q11	aux + neg + $NP aux|x~neg|not + $NP	Q6, Q7, Q8	Isn't he a neat dragon? Why didn't you say so?		different Q

Notes:

Q4: count wh + v:cop ONLY as second point, unless V4 has two points

Sentence Structure Points

Level	Include	Also	Example	Exclude	2nd point
S1	x + x (any 2 words)		here hammer		1 stem diff
S2	$NP + v	S1	Mommy fell.		all different
S3	v + $NP	S1	help me,		all different
S4	$NP + v + $NP	S2, S3	I need that		all different
S5	conj|and conj|or				stem different
S6	v + v	S1	I like play. You saw fall.	wanna play	different VP
S7	x + and + x x + or + x	S1, S5	red and blue, you or me		different phrases
S8	inf + (X) + v	S6, V5	I like to swim. Want to feed her?	see below	different verb
S9	let/make/help/watch + v	S6	let's play this. help it stand. make her quiet	make it pretty (no second verb)	different conj
S10	conj	S5	but, because, after, since	and, or, then (those are S5)	different conj
S11*	v + COMP	S6	I know you broke it.		different clause
S12*	S + and + S	S5, S6	I start it and you finish it.		different clauses
S13*	v + wh clause	S6, S10	Here is where it hurts. Tell me if it hurts.		different clause
S14*	OBJ + IOBJ	S3	Read me the book. Read the book to/for me.	Here's one for you. Gimme that.	different phrases
S15	v + (X) + v + (X) + v	S6	She told them to hurry and run.		different verbs
S16*	relative clause	S6	the one that fits		diff clause
S17*	COMP new subject	S8	I need you to help		diff clause
S18	n:gerund		This is for hammering		diff clause
S19*	Subject Clause	S6	The one I like best is		diff clause
S20*	passive, tags	S11	This guy is killed by the dragon.		diff clause

Notes:

S8 excludes: wanna feed her (V5 only); I like swim (S6 only); to ride on (no main verb)

S11, 12, 13, 14, 16, 17, 19, 20 require use of the %gra line.

S20 is ill-defined

Cascading credits for S1 are not computed, because children always get full on S1.

8.7.4 Unique Options

IPSYN has the following unique options:

+cN: analyse N complete unique utterances. (default: 100 utterances)

+d : do not show file and line number where points are found.

+lF: specify ipsyn rules file name F

8.8 KIDEVAL

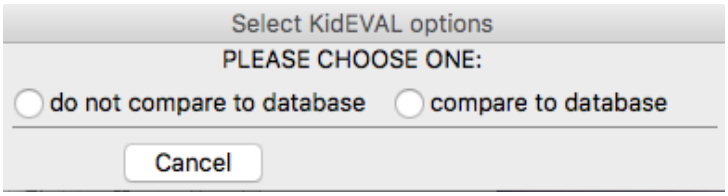
KIDEVAL is a program that provides automatic analysis of a language sample that has been transcribed in the CHAT format. Using various components of the CLAN program, KIDEVAL automatically computes these measures, which are entered into a spreadsheet. The KIDEVAL spreadsheet includes columns for each measure. If the analysis operates on multiple transcripts from the same or different children, each transcript will have values in its own row.

8.8.1 KIDEVAL for summaries

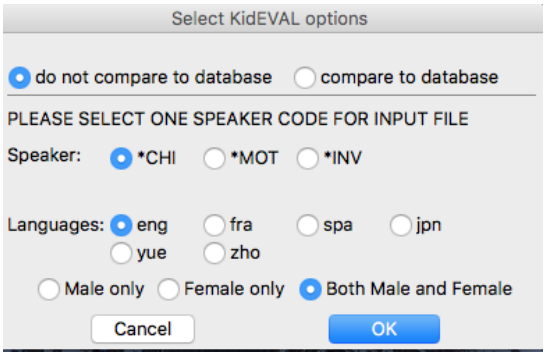
KIDEVAL performs analyses like those of EVAL, but in forms that are better suited to the analysis of child language data. It combines the various measures computed by DSS, FREQ, MORTABLE, MLU, TIMEDUR, and VOCD into a single analytic report. Unlike EVAL, KIDEVAL uses a configuration file that allows users to tailor analyses for their problems and interests.

Users may note that the numbers produced by KIDEVAL analyses often do not match those produced by separate runs of FREQ, MLU, and VOCD. The reasons for this depend on properties of each program. In MLU "Number of Utts" refers to MLU utterances which in KIDEVAL is called "MLU Utts". For FREQ, the default version of the program counts surface word forms on the main line. This means that, without additional switches, FREQ counts full words such as *dog* and *dogs* on the main line as two different word types. However, KIDEVAL bases its count of word types on lemmas on the %mor tier. On that line, *n|dog-PL* and *n|dog* both have the same lemma or word type. VOCD uses a random number generator, so if you run the same VOCD command twice you will get two different results. Every time you run VOCD on the same data you will get slightly different results.

To run KIDEVAL, you can make use of CLAN's dialog facility. Use the **Progs** button to select KIDEVAL and then click the **Option** button. You will be asked to choose the files you wish to analyze. For example, you could navigate to the /examples/transcripts/ne32 folder and then choose to add all five files and click **Done**. You will then see this dialog:



If you select "do not compare to database" the dialog will change to this form:



If you select the options displayed above, CLAN will compile and use this command:

```
kideval @ +leng +t*CHI:
```

The result will be sent to an Excel file, as noted in CLAN's output window with this line:

```
Output file </Applications/CLAN/examples/ne32/14.kideval.xls>
```

You can triple-click on that line, and it will fire up Microsoft Excel. Excel will ask you if you really want to open the file and just say "yes". The output (with most columns removed to fit on this page) will look like this:

A	B	C	D	E	F	G	H	I	J
File	Language	Custom_field	Total Utts	MLU Utts	MLU Words	MLU Morphe	MLU100 Utts	MLU100 Wo	MLU100 Mo
55.cha	eng		145	135	2.259	2.444	100	2.25	2.42
66.cha	eng		106	102	2.637	2.961	100	2.61	2.94
68.cha	eng		118	106	3	3.453	100	2.93	3.37
98.cha	eng		218	216	1.481	1.597	100	1.36	1.47

Using a similar process for Mandarin Chinese, we can select all the files in the Tong corpus in Chinese and the result (with many columns removed) will be:

	A	B	C	D	E	F	G	H	I	J	K	L
1	File	Language	Age(Month)	Total Utts	MLU Utts	MLU Words	MLU Morphe	MLU100 Utts	MLU100 Wo	MLU100 Mo	FREQ types	FREQ token
2	130202.cha	zho	20	103	101	1.644	1.644	100	1.63	1.63	55	17
3	130309.cha	zho	21	437	428	2.36	2.371	100	2.6	2.6	133	102
4	130405.cha	zho	22	433	406	2.599	2.621	100	2.86	2.91	188	111
5	130504.cha	zho	23	388	382	2.725	2.775	100	2.68	2.71	233	105
6	130607.cha	zho	24	450	445	2.811	2.818	100	2.82	2.82	207	126
7	130705.cha	zho	25	377	375	2.963	2.963	100	2.45	2.45	196	111
8	130802.cha	zho	26	510	507	2.923	2.949	100	2.56	2.59	247	149
9	130901.cha	zho	27	294	294	3.585	3.612	100	3.93	3.96	173	105
10	130929.cha	zho	27	428	426	3.662	3.723	100	3.48	3.53	248	156
11	131103.cha	zho	29	490	489	3.975	4.01	100	3.59	3.62	265	194
12	131215.cha	zho	30	399	395	3.096	3.127	100	3.03	3.05	230	123
13	131229.cha	zho	30	398	398	3.945	3.972	100	3.36	3.39	287	157
14	140203.cha	zho	32	454	454	3.198	3.256	100	3.12	3.21	253	145
15	140223.cha	zho	32	446	442	3.572	3.597	100	3.63	3.66	281	159
16	140323.cha	zho	33	303	302	3.401	3.467	100	3.26	3.29	210	103
17	140423.cha	zho	34	529	525	3.724	3.771	100	2.97	3	285	196
18	140525.cha	zho	35	454	453	3.733	3.746	100	4.24	4.25	279	169
19	140628.cha	zho	36	466	465	4.133	4.196	100	3.87	3.88	294	193
20	140726.cha	zho	37	455	454	3.839	3.861	100	3.62	3.67	248	175
21	140824.cha	zho	38	408	408	2.662	2.672	100	3.1	3.11	214	108
22	140921.cha	zho	39	497	495	4.59	4.622	100	4.86	4.9	363	228
23	141025.cha	zho	40	470	467	3.623	3.64	100	4.04	4.06	307	169

Here is a description of the fields that will be computed in these outputs for English:

1. Total Utts: total utterances,
2. MLU Utts: number of utterances, as used for computing MLU,
3. MLU Words: MLU in words,
4. MLU Morphemes: MLU in morphemes,
5. MLU 100 Utts: MLU of the first 100 child utterances in morphemes,
6. MLU 100 Words: MLU of the first 100 child utterances in words,
7. MLU 100 Morphemes: MLU of the first 100 child utterances in morphemes,
8. FREQ types: total word types, as used for computing FREQ
9. FREQ tokens: total word tokens,
10. FREQ TTR: type/token ratio,
11. NDW 100: number of different words in the first 100 words in the sample,
12. VOCD score: KIDEVAL will warn if the sample is too small to compute VOCD,
13. Verbs/Utt: verbs per utterance. This can be less than 1.0 for young children,
14. TD Words: total number of words for each speaker, as used for TIMEDUR
15. TD Utts: total number of utterances for each speaker (no exclusionary criteria),
16. TD Time: total duration in seconds of utterances for each speaker,
17. TD Words/Time: words per second,
18. TD Utts/Time: utterances per second,
19. Word Errors: number of words involved in errors,
20. Utt Errors: number of utterances involved in errors,
21. Retracing [//]: number of retracings,
22. Repetition [/]: number of repetitions,

23. DSS Utterances: number of DSS-eligible utterances (default 50 required),
24. DSS: Developmental Sentence Score,
25. IPSyn Utterances: number of IPSyn-eligible utterances (default 50 required),
26. IPSyn Total
27. MOR words: the number of words according to the %mor tier
28. The frequencies of each of Brown's 14 grammatical morphemes, which are:

a.	-PRESP	the present participle -*ing*, as in *swimming.*
b.	in	the preposition in, as in the cheese is in the bag.
c.	on	the preposition *on,* as in *put it on*.
d.	-PL	the regular plural, as in *dogs*.
e.	&PAST	the irregular past, as in *fell*.
f.	~poss	the possessive clitic, as in *John's*
g.	Cop	the uncontractible copula as in *Is Meg nice? Meg is*.
h.	det:art	the determiner, as in *the ball*.
i.	-PAST	the regular past, as in *jumped*.
j.	-3S	the regular third person singular present, as in *runs*.
k.	&3S	the irregular third person singular present, as in *does* or *has.*
l.	aux	the uncontractible auxiliary, as in *Is John running? Yes, he is*.
m.	~cop	the contracted copula, as in *Meg's tall.*
n.	~aux	the contracted auxiliary, as in *John's going*.

For some of these categories, there are several possible morphemes in the category. So, det:art can be either *a, an,* or *the*; aux can be any of the forms of the verbs *to be, to have,* or *to get*; and cop can be any of the forms of the verbs *to be* and *to become*, and certain uses of *look, seem,* and *stay.*

29. Total non-zero MOR: This is the count of the number of Brown's morphemes that had at least one instance. For example, if 10 of the 14 morphemes appeared, this number would be 10.

The first 22 of the measures in KIDEVAL are the same for all languages. Fields 23 and 24 are only computed for English and Japanese, because only those two languages have DSS. Also, fields 25 and 26 are only meaningful for English, since IPSyn only exists so far for English. For the many indices list as parts of field 28, the actual morphemic forms that are being tracked vary totally from language to language. The best way to follow these is to run KIDEVAL on a single file and then to look at the column headers for each field after *mor_Words.* We can modify this set by adding or removing morphemes, based on user requests.

To run KIDEVAL, a transcript must have at least 50 utterances. If the transcript does not contain enough utterances for a given index, N/A (not applicable) is inserted. KIDEVAL now uses the new smaller rule set recommended by Yang et al., as described in the section on IPSyn. For the MLU computation, sentences marked by [+ mlue] are excluded from the analysis. For the DSS computation, sentences marked by [+ dsse] are excluded. For the IPSyn computation sentences marked by [+ ipe] are excluded. If a transcript does not have time values entered, then columns 15-18 will not be meaningful.

8.8.2 Creating a custom language script

You can either use built-in KIDEVAL indices or you can configure your own set of indices. If you wish to use KIDEVAL to compare a file with the larger database, you must use the built-in index files and the built-in commands.

However, if you need to track a different set of indices, you may want to develop your own custom language file. If you do this, then the +l switch should use the name of the language file you created which must be put into the CLAN/lib/kideval folder. For example, for Dutch, it should have the name nld.cut. When you use an external language file, then you can only use KIDEVAL to compute summaries and not to compare with a database.

It is possible to control the operation of KIDEVAL by editing the script files for individual languages that are stored in the /lib/kideval folder. To use one of these files, just add its name to the +l switch, as in +lfranew.cut. If you use the switch +leng or +lfra, then the program relies on a built-in script file, instead of your user-defined file.

Within the script file, each line defines a type of search string. For example, this line searches for all instances of masculine singular marking in French:

```
+&m,&sg +&m,-SG  "m-sg"
```

Items separated by a comma are treated as AND; items separated by a space are treated as OR. To include combinations of morphemes in a KIDEVAL spreadsheet, you must run a separate FREQ program, such as this one that looks for adj+noun or noun+adj combinations in French:

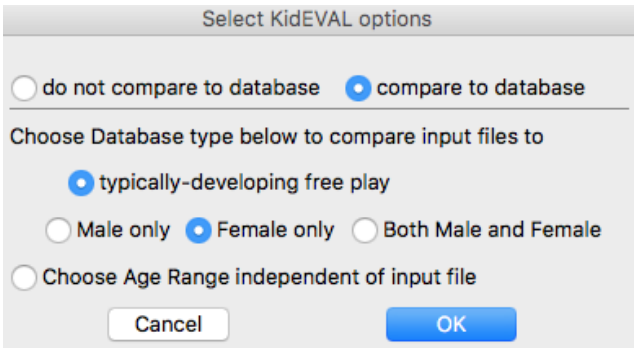
```
freq +sm"|adj |n" +sm"|n |adj" +d2 *.cha
```

This command will create an Excel output structured like that for KIDEVAL and you may wish to cut and paste the relevant columns from that output into your overall KIDEVAL spreadsheet.

8.8.3 KIDEVAL with a comparison database

In addition to providing summary profiles for a single file or a collection of files, KIDEVAL can compare results for a single file with results from a large comparison database of CHAT transcripts. This comparison then assigns a standard deviation score with significance levels to each score in the transcript being compared. By default, the comparison is against samples of children in the same 6-month age range.

Currently, comparison databases are available for English (ENG), Japanese (JPN), and Mandarin (ZHO). Before beginning a comparison database analysis, you should select "Get KIDEVAL Database" from CLAN's **File** menu. It will download to the user library preference folder on your computer. If you use the KIDEVAL dialog to select a comparison database, the dialog will change to this form:



Now, the dialogs will be creating this CLAN command:

```
kideval @ +LinkAge +leng +t*CHI: +dfp~"2;6-2;11|female"
```

Comparisons involve comparing a single transcript with the overall comparison database. For example, if we compare the barry.cha file in the CLAN examples folder with the English database, we will get this output (with many columns removed):

	A	B	C	D	E	F	G	H	I	J	K	L
1	File/DB	Language	Corpus	Total Utts	MLU Utts	MLU Words	MLU Morph	FREQ types	FREQ tokens	FREQ TTR	NDW/100	VOCD D_opt
2	barry.cha	eng	samples	79	77	3.39	4.169	82	305	0.269	42	33.97
3	+/-SD	.	.	-0.995	-0.973	-0.161	0.17	-1.495	-0.974	0.471	-1.006	-1.95
4								*			*	*
5	Mean Databa	.	.	318.467	303.008	3.541	3.985	220.444	1218.335	0.229	49.377	69.461
6	SD Database	.	.	240.699	232.243	0.94	1.084	92.603	937.303	0.084	7.334	18.194
7	Number files	.	.	257	257	257	257	257	257	257	257	257
8	+/- SD * = 1 SD, ** = 2 SD											
9	Database keywords: 3;-3;5											

In this output, the asterisks in row 4 indicate that this sample differs significantly from the comparison group on several measures. The corpora that are used for the English-NA KIDEVAL comparison databases are

Bates/Free20	Bates/Free28	Bernstein/Children
Bliss	Bloom70	Bloom73
Braunwald	Brown	Clark
Demetras1	Demetras2	Feldman
Gathercole	Gleason/Father	Gleason/Mother
Hall	Higginson	HSLLD/HV1/TP
HSLLD/HV2/TP	HSLLD/HV3/TP	MacWhinney
McCune	NewEngland	Post
Providence	Sachs	Snow
Suppes	Tardif	Valian
VanHouten	VanKleeck	Warren
Weist		

The corpora used for Mandarin are Tong and Zhou1. All French corpora are used for French.

8.8.4 Unique Options

KIDEVAL has the following unique options:

+bS: add S characters to the morpheme delimiters list (default: -#~+)
-bS: remove S characters from the morphemes list (-b: empty morphemes list)
+d : debug
+g: gem tier should contain all words specified by +gS
-g: look for gems in database only
+gS: select gems which are labeled by label S
+lF: specify language script file F
+n: gem is terminated by the next @G (default: automatic detection)
-n: gem is defined by @Bg and @Eg (default: automatic detection)
+o4: output raw values instead of percentage values

8.9 MORTABLE

MORTABLE uses the %mor line to create a frequency table of parts of speech and affixes in a format that can be opened directly in Excel. The command line needs to include the script file which is in the CLAN/lib/mortable folder, for example:

```
mortable +t*PAR +leng *.cha
```

Columns M-AF provide the percentage of each part-of-speech — e.g., adjectives, adverb, auxiliaries, conjunctions. The script for these percentage calculations uses an "OR" format, so that the data in each column is mutually exclusive. Columns AE-AS provide the percentages of each affix. These are calculated in a non-exclusive fashion.

If you want the actual count of the items found instead of percentages, add +o4 to the command line. If you want MORTABLE to automatically calculate cumulative totals of parts-of-speech, you can make modifications to the eng.cut file found in CLAN/lib/mortable. Here is an example of how to do this.

```
AND
# +|v,|v:*       "% v,v:*"
```

If you wanted a cumulative total of all pronouns (including pro:indef, pro:per, pro:wh, pro:refl, pro:poss, and pro), you could enter the following into the script file under the AND section and you would see a column in your spreadsheet called "pro-total":

```
+|pro,|pro:*     "pro-total"
```

For cases where target replacements are in the transcript next to error productions with missing morphemes (e.g., he is kick [: kicking] [* m:0ing] the ball), the EVAL and MORTABLE programs will reflect the speaker's morphological production (e.g., v|kick) and not count anything that was not produced (e.g., part|kick-PRESP). For cases where target replacements are in the transcript next to error productions for superfluous morphemes (e.g., there is one birds [: bird] [* m:+s] in the tree), the EVAL and MORTABLE programs will not count the superfluous morphemes (e.g., n|bird-PL) because they were not used correctly.

8.10 SUGAR

This program computes the SUGAR (Sampling Utterances and Grammatical Analysis Revised) profile (Pavelko & Owens, 2017). For a discussion of problems with SUGAR see (Guo, Eisenberg, Bernstein Ratner, & MacWhinney, 2018). For SUGAR, the corpus of utterances includes the first 50 utterances. The basic format of the command is:

sugar +t*CHI filename.cha

There are just four metrics to be computed:

1. MLU-S: This measure is the same as the MLU currently in CLAN.
2. TNW: This counts the total number of words in the 50 utterances.
3. WPS: This computes (number of words) / (number of sentences). For this, each utterance with a verb counts as a sentence.
4. CPS: This computes (number of clauses) / (number of sentences). For this, each sentence counts as one clause and then each of the GRs marking embedded structures in the %gra line (section 7.8.14 of the CLAN manual) counts as an additional clause.

9 Format Conversion Commands

CLAN includes several commands that convert transcripts from other formats to CHAT and from CHAT to these other formats. These commands convert to other formats:

- CHAT2ANVIL
- CHAT2CA
- CHAT2CONLL
- CHAT2CA
- CHAT2ELAN
- CHAT2PRAAT
- CHAT2SRT
- CHAT2TEXT
- CHAT2XMAR

These commands convert from other formats to CHAT.

- ANVIL2CHAT
- CONLL2CHAT
- ELAN2CHAT
- LAB2CHAT
- LENA2CHAT
- LIPP2CHAT
- PRAAT2CHAT
- RTF2CHAT
- SALT2CHAT
- SRT2CHAT
- TEXT2CHAT

Descriptions of each of these commands follow here alphabetically in this section.

9.1 CHAT2ANVIL

This program converts a CHAT file to ANVIL format.

9.2 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 speedup and slowdown are marked by inward and outward pointing arrows.

9.3 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. The conversion of CHAT files to ELAN is fairly straightforward. You just run this command in the CLAN Commands window:

```
chat2elan +emp4 *.cha
```

The +e switch is used to specify the media file type, which could be mp4, mov, wav, or mp3. CHAT main tiers appear in ELAN with the speaker's name, such as *BET which becomes "BET". Dependent tiers, which are called "child tiers" in ELAN, are then coded as owned by a speaker, as in gpx@BET for the %gpx tier linked to the BET speaker.

9.4 CHAT2PRAAT

This program converts a CHAT file to the Praat format. When running this, you need to add the file type of the audio using the +e switch as in +emp3.

9.5 CHAT2SRT

This program converts a CHAT file to SRT format for captioning video. The use of this program is described in a screencast available from http://talkbank.org/screencasts. On the Mac, you will need to purchase Subtitle Writer for $4.99 from the App Store.

1. Run this command: chat2srt shoes.cha to produce shoes.srt
2. Open Subtitle Writer.
3. Click **Add** and add shoes.srt
4. Select the language as **English**.
5. Click **Import Movie** and select shoes.mp4
6. Click Save Subtitled Video
7. Open and play the resultant captioned movie.

If you want to use the English gloss in the %glo line in a file instead of the main line for the captions, then use this command: chat2srt +t%glo shoes.cha to produce the .srt file.

9.6 CHAT2TEXT

This program converts a CHAT file to a series of text lines for analysis by concordance programs such as AntConc. This command is implemented as a simple alias based on the FLO command: flo +cr +t* The FLO program with the +cr switch removes all the various markup of CHAT.

9.7 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/.

9.8 ANVIL2CHAT

This program converts an ANVIL file to a CHAT file. For conversion in the opposite direction, you can use CHAT2ANVIL

9.9 ELAN2CHAT

This program converts ELAN files to CHAT files. Use CHAT2ELAN for conversion in the opposite direction. The command is just: ELAN2CHAT filename.cha. If your file began in CHAT, then the tiers probably already have names that will pass on without problems during the conversion. However, if your file was produced originally in ELAN, it may be necessary to rename the tiers to CHAT format before running elan2chat. There are basically two different pathways for conversion of ELAN to CHAT.

CHAT-compatible ELAN files: The first pathway is the simplest. In this method, you create ELAN files from scratch that maximize CHAT compatibility. To do this, you give the main tier a 3-letter name, such as BET in the screencast example. Then, the %gpx dependent tier for that speaker is named gpx@BET and it should be a child under the BET top level tier. You should avoid use of **Time Subdivision** and **Symbolic Subdivision** tier type. Instead, please make use of the **Included In** tier type.

ELAN files that must be reorganized: In the second pathway, you would be working with ELAN files that had not been structured from the beginning to maximize CHAT compatibility. In that case, there are ways to reorganize the ELAN files to increase compatibility. The steps are:

1. Renaming tiers. For this, you need to select one tier as the top-level tier. Usually this would be the main speaker tier(s) with one such tier for each speaker.
2. Then you would align the child or dependent tiers with the top level or main tiers. If you are coding three types of dependent information and you have three speakers, you would then end up with 9 tiers.
3. It is also possible to have tiers that are children of child tiers, if they are fully aligned. However, it is not possible to go further into embedding with a third level of dependency.
4. If a corpus has a mix of aligned and unaligned annotations on a **Symbolic Subdivision** tier, you should either convert the tier to **Included In** or make sure that all annotations are aligned.
5. Finally, you may wish to remove some tiers that don't adapt well to the CHAT structure or which are not important for your general analysis.

When reorganizing ELAN tiers, you need to consider these operations:

1. Changing the hierarchy of tiers in ELAN is only possible via a copy operation (there is an option **Reparent Tier** but that also creates a copy of the tier). If the result is acceptable, the original tier can be removed. It is implemented this way because changing the hierarchy can involve changing the type of the tier and, therefore, the constraints applied to annotations (annotations can be removed or concatenated etc. by this operation).
2. Renaming of tiers can be applied to a set of files / a corpus (File->Multiple File Processing->Edit Multiple Files...). This (mainly) makes sense if there is some consistency of tier names in that corpus/set of files.

9.10 LAB2CHAT

This program converts files in WaveSurfer format, such as the files from Stockholm

University to CHAT format. Each utterance is on a separate line and the line begins with the begin and end times in seconds. The WaveSurfer or LABB format has separate files for each speaker, so there must be an attribs.cut file, like the one in /lib/fixes/wavesurf-attribs.cut to declare who is who, as in:

```
;"Chat Speaker"  "Original tag litman"
*MOT          mamman            Mother
*CHI          barnet            Target_Child
```

9.11 LENA2CHAT

This program is designed to convert LENA *.its files into CHAT format. It assumes that all the *.its files for a given child, along with the *.wav audio files, are collected into a single folder that has a 3- or 4-character name, such as AB04 or 0134. The command of "lena2chat *.its" is issued from within the folder. After running the command, please create a new folder inside this folder called 0its and put the *.its files into that folder. The *.cha files created by the program stay at the top level and they will be given names based on the folder name and the age of the child. The .wav files will also be renamed in this way. So, if the child's age is 2;04.05 and the folder name is AB04, then the file will be named AB04_020405.cha and the corresponding audio file will be named AB04_020505.wav. The @Media line will use the new name of the media file, but the original name of the media file will be preserved in a line such as:

```
@Comment: old media file name is e20100728_143446_003489
```

The program is designed to convert whole folders at a time, rather than single files, although a folder with only one file will also be converted. To convert a series of folders at once, you can add the +re switch. So, conversion of a whole corpus would use this form of the command:

```
lena2chat +re *.its
```

9.12 LIPP2CHAT

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.

9.13 PRAAT2CHAT

This program converts files in the PRAAT format to files in CHAT format. The /examples/praat2chat folder contains a 0readme.txt file that explains how to use the program. The following material is the same as what is given in that file:

The files in this folder illustrate how to convert Praat files to CHAT and vice versa.

The simplest case arises when you first convert a CHAT file like clip.cha to Praat, you use this command:

```
chat2praat clip.cha +emp3
```

The +e switch specifies the format of the audio as either mp3 or wav. The output of this command is clip.c2praat.textGrid. You can convert this back to CHAT format using this command:

```
praat2chat clip.c2praat.textGrid
```

In this case, no attribs.cut file is needed, because the orginal file was already in CHAT format. The output of this command is clip.xh2praat.praat.cha. Note that this file is identical to clip.cha.

However, if you begin with a Praat file that was never converted to CHAT, you will need to create an attribs.cut file and you use this command:

```
praat2chat -opcl +dattribs.cut praat.textGrid
```

This command uses the declarations in the attribs.cut file illustrated in this folder and produces praat.praat.cha as output. The +d swtich with an attribs.cut file is only needed when converting from a Praat file created originally inside Praat, not when the Praat file was created from CHAT, and not when converting from CHAT to Praat. The attribs.cut file tells the program which Praat tag gives the Speaker role and which ones give dependent tiers. After running praat2chat, you will need to fix the headers, run DELIM to add periods for utterances and perform various similar operations required by CHAT format.

9.14 RTF2CHAT

This program is used to take data that was formatted in Word and convert it to CHAT.

9.15 SALT2CHAT

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, SALT2CHAT sends its output to a file. Here is the most common use of this program:

```
salt2chat file.cut
```

It may be useful to note a few details of the ways in which SALT2CHAT operates on SALT files:

1. When SALT2CHAT 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 SALT2CHAT runs. The CHSTRING program can be used to insert + in the desired codes or in all codes, if required.
4. Times in minutes and seconds are converted to times in hours:minutes:seconds.
5. A %def tier is created for coding definitions.

9.15.1 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 SALT2CHAT 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.

After running SALT2CHAT, you will probably want to use the LOWCASE program to recapitalize proper nouns.

9.16 SRT2CHAT

This program converts files in SRT format to CHAT format. For the file headers, it uses TXT as the speaker and English as the default language. Times are encoded in the bullets. After conversion, you may want to put the individual lines into full CHAT format. Alternatively, you can insert the @Options: heritage line to preserve the initial transcripts and timing.

9.17 TEXT2CHAT

The TEXT2CHAT program is quite simple. It takes a set of sentences in paragraph form and converts them to a CHAT file. Blank lines are treated as possible paragraph breaks and are noted with @Blank headers. To illustrate the operation of TEXT2CHAT, here are the results of running TEXT2CHAT on the previous three sentences:

```
@Begin
@Languages:   eng
@Participants: TXT Text
@ID:    ent|text|TXT|||||Text|||
*TXT:   the text2chat program is quite simple.
*TXT:   it takes a set of sentences in paragraph form and
   converts them to a chat file.
*TXT:   blank lines are considered to be possible paragraph
   breaks and are noted with @blank headers.
@End
```

Problems can arise when there is extraneous punctuation in the original, as in forms such as *St. Louis* which would generate a new line at *Louis.* To avoid this, one can either remove these problems beforehand or in the resultant CHAT file.

10 Reformatting Commands

These commands are useful when a researcher wants to add features to files that have already passed CHECK and are in good CHAT format. Some of these add additional material to files; others move around information already in the files. Clicking in the page number will take you to the relevant command.

Command	Page	Function
CHSTRING	161	Changes one string to another, often using a changes file.
DATES	163	Given a birthday and current data, computes the age.
FLO	163	Adds a simplified %flo line to each line in a transcript.
INDENT	164	Realigns overlaps in CA transcripts.
LONGTIER	164	Makes each utterance one line long.
REPEAT	164	Marks repeated sequences of words as repetitions.
RETRACE	164	Marks retraced segments.
TIERORDER	165	Places all dependent tiers into a user-specified order.
TRIM	165	Remove specified dependent tiers.

10.1 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"

You must use the quotation marks to surround the two strings. 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 several 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, using this command:

```
chstring +cbullets.cut *.cha
```

10.2 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;03.01 +b 12-jan-1962
```

you should get the following output:

```
@Age of Child: 2;03.01
@Birth of Child: 12-JAN-1962
@Date: 13-APR-1964
```

You can also use the date format of MM/DD/YY, as in this version of the command:

```
dates +b 08/31/63 +d 07/30/64
```

If your files have the child's age in the @ID header, and if you know the child's date of birth, but do not have the @Date field, you can create a set of new files with the @Date information, using this version of the command:

```
dates +bCHI 08/31/63 *.cha
```

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.

10.3 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.

10.4 INDENT

This program is used to realign the overlap marks in CA files. The files must be in a fixed width font such as CAFont.

10.5 LINES

This program inserts line numbers that can be saved when closing the file, based on the numbering system used by the "Show Line Numbers" option in the Mode menu. Apart from the options available to other programs, LINES uses the +n option to remove all the line/tier numbers.

10.6 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 for cleaning up files, because it eliminates having to think about string replacements across line breaks.

10.7 MEDIALINE

This program inserts the @Media field, based on the name of the transcript, along with options for declaring a=audio, v=video, m=missing, and u=unlinked. So, this command

```
medialine +a +u test.cha
```

will insert this line in the test.cha file:

```
@Media:   test, audio, unlinked
```

10.8 REPEAT

REPEAT if two consecutive main tiers are identical then the postcode [+ rep] is inserted at the end of the second tier.

10.9 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 the reformatted material and no additional %ret tier is created.

10.10 SEGMENT

This command is used to modify the utterance segmentation produced in the Batchalign pipeline for automatic speech recognition (https://github.com/talkbank). Once Batchalign has finished, you can use continuous playback (Esc-8) to play through the transcript to make sure that words are correctly recognized and that utterances are properly segmented. If one utterance should be joined to the following utterance, you can add &&& at the end of the line after the time bullet and remove the terminator on the first utterance. If an

utterance should be broken up, then you can enter &&& at the break location preceded by the terminator needed for the first utterance. When doing this, SEGMENT assumes that there are no additional dependent tiers other than %wor. So, if there are %mor and %gra tiers, you can use TRIM to remove them and then rerun MOR after completing SEGMENT.

10.11 TIERORDER

TIERORDER puts the dependent tiers into a consistent alphabetical order. The /lib/fixes/tierorder.cut file can be used to control this order by rearranging the order of tiers in that file.

10.12 TRIM

This command is designed to allow you to remove coding tiers from CHAT files. For example, to remove all the %mor lines from files without changing anything else, you can use this command:

```
trim -t%mor *.cha +1
```

This command is equivalent to this more verbose KWAL command:

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

To remove all dependent tiers, you can use this command:

```
trim -t%* *.cha +1
```

11 Format Repair Commands

These commands are used to repair the format of files that are not yet in the correct current CHAT format. These could either be files that were created by older versions of CLAN or files that were not run through CHECK during their creation. Researchers who are using current versions of CLAN and who have files that are passing CHECK do not need to use any of these commands.

11.1 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.

11.2 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, unless you add the +d switch and you are sure that the line in question is not already in UTF. The +c switch uses the unicode.cut file in CLAN/lib/fixes 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.

+b : add BOM symbol to the output files
+cN: specify column number (3-7) (default: 4, IPATimes)
+d: convert ONLY tiers specified with +t option
+d1: remove bullets from data file
+d2: add BOM encoding information at the beginning of a CHAT file to help applications, such as NVivo or MS-Word, to read it better
+oS: specify code page. Please type "+o?" for full listing of codes
utf16 - Unicode UTF-16 data file
macl - Mac Latin (German, Spanish ...)
pcl - PC Latin (German, Spanish ...)

11.3 DELIM

DELIM inserts a period at the end of every main line if it does not currently have one. To do this for all tiers use the +t* switch.

11.4 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
+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"

11.5 FIXIT

FIXIT is used to break up tiers with multiple utterances into the standard format with one utterance per main line.

11.6 LOWCASE

This program is used to fix files that were not 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 with the name caps.cut and add the +d switch to avoid lowercasing these. You may also want to use the /lib/fixes/caps.cut file. It contains over 8000 proper nouns, many based on children's first names.

11.7 QUOTES

This program moves quoted material to its own separate tier.

12 Supplementary Commands

CLAN also includes these basic operating system type commands which allow you to manage file and folder structure.

12.1 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, such as commands.bat. Each line of that file is then executed as a CLAN command.

12.2 cd

This command will change your working directory. For example, is you have a subfolder called ne20, then the command cd ne20 will set your current working directory to ne20. To move up, you can type cd ..

12.3 dir

This command will provide a listing of the files in your current working directory.

12.4 info

Just typing "info" will list all the available CLAN commands.

12.5 ren(ame)

This command allows you to change file names in a variety of ways. You can change case by using -u for upper and -l for lower. The -c and -t switches allow you to change the creator signature and file types recognized by Macintosh. The –f switch forces file replacement. Here are some examples:

Rename a series of files with names like “child.CHA (Word 5)”:

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

Rename the output from FREQ and replace the original files:

```
ren –f *.frq.cex *.cha
```

Taking out spaces (because CLAN has trouble with spaces in names):

```
ren'* -e.cha' '*-e.cha'
```

If you have two variables in your input, you can control which is replaced by adding either \1 or \2 to the output. So, if you use \1, then the first variable is preserved, and if you use \2, then the second variable is preserved. So, if the input is aki12_boofoo.cha, then this command creates 12.cha as output:

```
ren aki*_*.cha \1*.cha
```

and this command creates boofoo.cha as output:

```
ren aki*_*.cha \2*.cha
```

12.6 rm

This command will delete files. It can be used with wild cards and recursively, as in

```
rm -re *.mor.cex
```

It is important to be careful with the use of this command. A good practice is to make a copy of the folder(s) to which it applies before running the command.

13 Options

This chapter describes the various options or switches that are shared across the CLAN 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.

13.1 +F Option

This option allows you to send output to a file rather than to the screen. By default, nearly all 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 SALT2CHAT. 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 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.

13.2 +K Option

This option controls case-sensitivity. A case-sensitive program is one that makes a distinction between uppercase and lowercase letters. Many of the CLAN commands are case-sensitive by default. If you type the name of each command, you will see a usage page indicating the default setting for the +k switch. Use of the +k option overrides the default state whatever that might be. 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.

13.3 +L Option

The +l option is used to provide language tags for every word in a bilingual corpus. Use of this switch does not actually change the file; rather these tags are represented in computer memory for the files and are used to provide full identification of the output of programs such as FREQ or KWAL. For examples of the operation of the +l switch in the context of the FREQ program, see the section of the FREQ program description that examines searches in bilingual corpora.

An additional variation on the +l switch is +l1 which serves to insert a language precode such as [- spa] for every utterance, including those that are unmarked without the use of this switch. If this switch is used, then it is possible to trace between language code-switching on the whole utterance level using commands such as the two following, where the first one tracks changes from French to Spanish and the second tracks changes from Spanish to French:

```
combo +b2 -l1 +s"\**:^[- fra]^*^\**:^[- spa]" *.cha
    combo +b2 -l1 +s"\**:^[- spa]^*^\**:^[- fra]" *.cha
```

13.4 +P Option

This switch is used to change the way in which CLAN processes certain word-internal symbols. Specifically, the programs typically consider compounds such as black+bird to be single words. However, if you add the switch +p+, then the plus symbol will be treated as a word delimiter. This means that a program like FREQ or MLU would treat black+bird as two separate words. Another character that you may wish to treat as a word separator is the underscore. If you use the switch +p_, then New_York would be treated as two words.

13.5 +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 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 are regarded as the same. If you use this switch, they are 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. The exception to this rule is for the WDLEN program. If you do not want this replacement, use this switch.

+r6 Retraced Material. By default, material in retracings is included in searches and counts. The exceptions are the EVAL, FREQ, MLT, MLU, and MODREP programs, for which retracings are excluded by default. The +r6 switch is used to change these default behaviors for those programs.

+r7 Do not remove prosodic symbols (/~^:) in words

+r8: Combine %mor tier items with replacement word [: …] and error code [* …] if any from speaker tier.

13.6 +S Option

This option allows you to search for a certain 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 +s@, as in +s@adverbs, which will search for the words in a file called adverbs.cut. If you want to use +s 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 matches to the +s string on the main line. If you want to include a search on other tiers, you need to add them with the +t switch. Also by default, unless you explicitly include the square brackets in your search string, the search will ignore any material that is enclosed in square brackets.

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 can be used with five types of material: (1) words, (2) codes or postcodes in square brackets, (3) text in angle brackets associated with codes within square brackets, (4) whole utterances associated with certain postcodes, and (5) particular postcodes themselves. Moreover, the switch can be used to include, exclude, or add information. The effect of the switch for the five different types across the three functions is described in the following three tables:

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 itself
<text>[code]	+s"<//>"	find only text marked by this code
utterance	+s"[+ imi]" +s"[- eng]"	find only text marked with this postcode or precode
postcode	+s"<+ imi>" +s"<- eng>"	find only this postcode or precode itself

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

Search Strings for Addition of Material 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

You can use either single or double quotation marks. However, for Unix and CLAN

commands on the web interface, you need to use single quotation marks. When your search string does not include any metacharacters or delimiters, you can omit the quotation marks altogether.

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 +t%cod
```

and your text has these codes

```
$gf $gf:a $gf:b $gf:c
```

your output will be

```
$gf           3
$gf:a         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
```

13.7 +T Option

This option allows you to include or exclude 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 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 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.

13.8 +U Option

This option merges the output of searches on 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 that set as a whole. For most commands, the switch merges all data for a given speaker across files. The commands that do this are: CHAINS, CHIP, COOCCUR, DIST, DSS, FREQ, FREQPOS, GEM, GEMFREQ, IPSYN, KEYMAP, MAXWD, MLT, MLU, MODREP, PHONFREQ, and WDLEN. There are several other commands for which there is a merged output, but that output separates data from different input files. These commands are COMBO, EVAL, KIDEVAL, KWAL, MORTABLE, TIMEDUR, and VOCD. If too many files are selected, CLAN may eventually be unable to complete this merger.

13.9 +V Option

This switch gives you the date when the current version of CLAN was compiled.

13.10 +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.

13.11 +X Option

This option is available in most of the analysis programs. It allows you to control the type and number of items in utterances being selected for analysis.

+xCNT: C (condition) can be greater than >, less than <, or equal = (>, <, =)
N (number) is the number of items to be included
T (type) is the type of item which can be words (w), characters (c), or morphemes (m). If "m" is used, there must be a %mor line.
+x<10c means to include all utterances with less than 10 characters
+x=0w means to include all utterances with zero words
+xS: include certain items in above count (Example: +xxxx +xyyy)
-xS: exclude certain items from above count
In the MOR and CHIP programs, +x has a different meaning.

13.12 +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. It also permits certain special operations on 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 their files to CHAT format using the TEXTIN program to avoid having to avoid use of the +y option.

If you want to search for information in specific headers, you may need to use the +y option. For example, if you want to count the number of utterances by CHI in a file, you can use this command:

```
freq +s"\*CHI" *.cha +u +y
```

13.13 +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 certain 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 first use KWAL with +z to extract the utterances you want and then run MLU on that output.

```
kwal +d +z50u +t*CHI sample.cha
kwal +sMommy sample.kwa.cex
```

If the +z switch specifies 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 also use postcodes to further control the process of inclusion or exclusion.

13.14 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, 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,” “cats’” and the contraction “cat’s”. 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,” “breaks,” and so forth.

The % character allows the program to match characters in the same way as the * symbol. 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 key-words, 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.

The underline character _ is like 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 in-serting 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.”

14 References

Aguado, G. (1988). Appraisal of the morpho-syntactic competence in a 2.5 month old child. *Infancia y Aprendizaje, 43*, 73-95.

Altenberg, E., Roberts, J., & Scarborough, H. (2018). Young children's structure production: A revision of the Index of Productive Syntax. *Language, Speech and Hearing Services in Schools, 49*(4), 1-14. doi:10.1044/2018_LSHSS-17-0092

Berndt, R., Wayland, S., Rochon, E., Saffran, E., & Schwartz, M. (2000). *Quantitative production analysis: A training manual for the analysis of aphasic sentence production*. Hove, UK: Psychology Press.

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, C., & Pressley, M. (1982). *Verbal processes in children*. New York, NY, NY: Springer-Verlag.

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.

Guo, L.-Y., Eisenberg, S., Bernstein Ratner, N., & MacWhinney, B. (2018). Is putting SUGAR (Sampling Utterances of Grammatical Analysis Revised) into Language Sample Analysis a good thing? A response to Pavelko and Owens (2017). *Language, Speech and Hearing Services in Schools, 49*(3), 622-627. doi:doi:10.1044/2018_LSHSS-17-0084

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., & Richards, B. (1997a). A new measure of lexical diversity. In A. Ryan & A. Wray (Eds.), *Evolving models of language* (pp. 58-71). Clevedon: Multilingual Matters.

Malvern, D., & Richards, B. (1997b, May 1997). *Trends in lexical diversity and type-token ratio in Templin: A mathematical modelling approach.* Paper presented at the poster presented at the 18th Annual Symposium on Research in Child Language Disorders, University of Wisconsin-Madison.

Malvern, D., Richards, B., Chipere, N., & Purán, P. (2004). *Lexical diversity and language*

development. New York, NY: Palgrave Macmillan.
Moerk, E. (1983). *The mother of Eve as a first language teacher*. Norwood, N.J.: 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, N.J.: Ablex Publishing Corporation.
Nice, M. (1925). Length of sentences as a criterion of a child's progress in speech. *Journal of Educational Psychology, 16*, 370-379.
Pavelko, S., & Owens, R. (2017). Sampling utterances abd grammatical analysis revised (SUGAR): New normative values for language sample analsysi measures. *Language, Speech, and Hearing Services in Schools*, 197-215.
Richards, B. J., & Malvern, D. D. (1996). Swedish verb morphology in language impaired children: Interpreting type-token ratios. *Logopedics Phoniatrics Vocology, 21*, 109-111.
Roberts, J., Altenberg, E., & Hunter, M. (2020). Machine-scored syntax: Comparison of the CLAN automatic scoring program to manual scoring. *Language, Speech and Hearing Services in Schools, 51*, 479-493.
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. (1990). Index of Productive Syntax. *Applied Psycholinguistics, 11*(1), 1-22. doi:10.1017/S0142716400008262
Scarborough, H., 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.
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. Retrieved from https://psyling.talkbank.org/years/1990/sokolov.pdf
Sokolov, J. L., & Snow, C. (Eds.). (1994). *Handbook of Research in Language Development using CHILDES*. Hillsdale, NJ: Erlbaum.
Templin, M. (1957). *Certain language skills in children*. Minneapolis, MN: University of Minnesota Press.
Thompson, C. K., Shapiro, L. P., Tait, M. E., Jacobs, B. J., Schneider, S. L., & Ballard, K. J. (1995). A system for the linguistic analysis of agrammatic language production. *Brain and Language, 51*, 124-129.
Wells, C. G. (1981). *Learning through interaction: The study of language development*. Cambridge: Cambridge University Press.
Yang, J. S., MacWhinney, B., & Ratner, N. B. (2021). The Index of Productive Syntax: Psychometric Properties and Suggested Modifications. *American Journal of Speech-Language Pathology*, 1-18.