The CHILDES Project

Part 2: The CLAN Programs

Brian MacWhinney
Carnegie Mellon University

February 22, 2016

Citation for last printed version:

1 Getting Started ................................................................................................................. 9
   1.1 Why you want to learn CLAN .................................................................................. 9
   1.2 Learning CLAN ......................................................................................................... 9
   1.3 Installing CLAN – Mac OS X .................................................................................. 10
   1.4 Setting the Download Directory – Mac OS X ....................................................... 10
   1.5 Installing CLAN – Windows .................................................................................... 10
   1.6 Setting the Download Directory – Windows ......................................................... 11
2 Using the Web ................................................................................................................... 12
   2.1 Community Resources ......................................................................................... 12
   2.2 Downloading Materials ....................................................................................... 12
   2.3 Using the Browsable Database .............................................................................. 13
   2.4 Downloading Transcripts and Media .................................................................. 13
3 Basic Commands Tutorial ................................................................................................. 15
   3.1 The Commands Window ....................................................................................... 15
   3.2 Typing Command Lines ....................................................................................... 19
   3.3 Sample Runs .......................................................................................................... 21
       3.3.1 Sample KWAL Run ..................................................................................... 21
       3.3.2 Sample FREQ Runs .................................................................................. 21
       3.3.3 Sample MLU Run ..................................................................................... 22
       3.3.4 Sample COMBO Run ................................................................................ 23
       3.3.5 Sample GEM and GEMFREQ Runs ........................................................... 23
4 Advanced Commands ...................................................................................................... 25
5 Exercises ........................................................................................................................ 30
   5.1 Contrasting Four Measures .................................................................................. 30
   5.2 MLU50 Analysis .................................................................................................... 31
   5.3 MLU5 Analysis ....................................................................................................... 33
   5.4 MLT Analysis ......................................................................................................... 34
   5.5 TTR Analysis ......................................................................................................... 35
   5.6 Generating Language Profiles ............................................................................. 35
   5.7 Further Exercises ................................................................................................... 37
6 The Editor ........................................................................................................................ 39
   6.1 The Interactive Tutorial ....................................................................................... 39
   6.2 Text Mode vs. CHAT Mode ................................................................................... 40
   6.3 File, Edit, and Font Menus .................................................................................... 40
   6.4 Default Window Positioning and Font Control .................................................. 40
   6.5 CA Styles ............................................................................................................... 41
   6.6 Setting Special Colors ......................................................................................... 41
   6.7 Searching ............................................................................................................... 41
   6.8 Hiding Tiers .......................................................................................................... 42
   6.9 Send to Sound Analyzer ...................................................................................... 42
   6.10 Tiers Menu Items ................................................................................................ 42
   6.11 Running CHECK Inside the Editor ...................................................................... 42
   6.12 Preferences and Options ..................................................................................... 43
   6.13 Coder Mode ......................................................................................................... 44
       6.13.1 Entering Codes ......................................................................................... 44
       6.13.2 Setting Up Your Codes File ..................................................................... 45
7 Media Linkage ........................................................................................................ 47
  7.1 Sonic Mode ....................................................................................................... 48
  7.2 Transcriber Mode ............................................................................................. 50
    7.2.1 Linking to an already existing transcript .................................................. 50
    7.2.2 To create a new transcript ........................................................................ 50
    7.2.3 Sparse Annotation ..................................................................................... 51
  7.3 Video Linking .................................................................................................... 52
  7.4 SoundWalker ..................................................................................................... 53
  7.5 Export to Partitur Editors ................................................................................ 54
  7.6 Playback Control .............................................................................................. 54
  7.7 Manual Editing ................................................................................................ 55
8 Features ................................................................................................................ 56
  8.1 Shell Commands ............................................................................................... 56
  8.2 Online Help ....................................................................................................... 57
  8.3 EMACS Commands Listing ............................................................................. 57
  8.4 Testing CLAN ................................................................................................... 57
  8.5 Bug Reports ...................................................................................................... 58
  8.6 Feature Requests .............................................................................................. 58
9 Analysis Commands Table .................................................................................. 59
  9.1 CHAINS ............................................................................................................. 60
    9.1.1 Sample Runs .............................................................................................. 60
    9.1.2 Unique Options ......................................................................................... 63
  9.2 CHECK .............................................................................................................. 64
    9.2.1 How CHECK Works .................................................................................. 64
    9.2.2 CHECK in CA Mode ................................................................................ 64
    9.2.3 Running CHECK ...................................................................................... 65
    9.2.4 Restrictions on Word Forms ..................................................................... 65
    9.2.5 Some Hints ............................................................................................... 66
    9.2.6 Unique Options .......................................................................................... 66
  9.3 CHIP .................................................................................................................. 67
    9.3.1 The Tier Creation System ...................................................................... 67
    9.3.2 The CHIP Coding System ...................................................................... 68
    9.3.3 Word Class Analysis ................................................................................ 69
    9.3.4 Summary Measures ................................................................................ 70
    9.3.5 Unique Options ......................................................................................... 72
  9.4 COMBO ............................................................................................................ 73
    9.4.1 Composing Search Strings ....................................................................... 73
    9.4.2 Examples of Search Strings ..................................................................... 74
    9.4.3 Referring to Files in Search Strings .......................................................... 75
    9.4.4 Cross-tier Combo ...................................................................................... 76
    9.4.5 Cluster Sequences in COMBO ................................................................ 76
    9.4.6 Tracking Final Words .............................................................................. 77
    9.4.7 Tracking Initial Words .............................................................................. 77
    9.4.8 Limiting with COMBO ............................................................................ 77
    9.4.9 Adding Codes with COMBO ................................................................... 78
    9.4.10 Unique Options ....................................................................................... 79
9.5 COOCUR ........................................................................................................... 81
  9.5.1 Unique Options ......................................................................................... 81
9.6 DIST ................................................................................................................... 82
  9.6.1 Unique Options ......................................................................................... 82
9.7 DSS ................................................................................................................... 83
  9.7.1 CHAT File Format Requirements .............................................................. 83
  9.7.2 Selection of a 50-sentence Corpus ............................................................. 83
  9.7.3 Automatic Calculation of DSS ................................................................. 84
  9.7.4 Sentence Points ......................................................................................... 84
  9.7.5 DSS Output ............................................................................................... 85
  9.7.6 DSS Summary ............................................................................................ 85
  9.7.7 DSS for Japanese ....................................................................................... 87
  9.7.8 How DSS works ......................................................................................... 89
  9.7.9 Unique Options ......................................................................................... 90
9.8 EVAL .................................................................................................................. 91
  9.8.1 Explanation of EVAL Measures ............................................................... 91
  9.8.2 EVAL Demo .............................................................................................. 92
  9.8.3 Unique Options ......................................................................................... 93
9.9 FREQ ................................................................................................................. 93
  9.9.1 What FREQ Ignores ................................................................................ 94
  9.9.2 Studying Lexical Groups using the +s@file switch ................................. 94
  9.9.3 Searches in Bilingual Corpora ................................................................. 94
  9.9.4 Building Concordances with FREQ ......................................................... 95
  9.9.5 Using Wildcards with FREQ ................................................................. 96
  9.9.6 FREQ for the %mor line ........................................................................ 98
  9.9.7 Errors relating to particular morphological codes ................................... 99
  9.9.8 Lemmatization .......................................................................................... 99
  9.9.9 Directing the Output of FREQ ............................................................... 100
  9.9.10 Limiting in FREQ ................................................................................ 101
  9.9.11 TTR for Lemmas .................................................................................... 102
  9.9.12 Studying Unique Words and Shared Words ........................................... 102
  9.9.13 Unique Options ....................................................................................... 103
  9.9.14 Further Illustrations ............................................................................... 105
9.10 FREQMERG ................................................................................................. 108
9.11 FREQPOS .................................................................................................... 109
  9.11.1 Unique Options ........................................................................................ 109
9.12 GEM ............................................................................................................... 110
  9.12.1 Sample Runs ............................................................................................ 110
  9.12.2 Limiting With GEM .............................................................................. 111
  9.12.3 Unique Options ....................................................................................... 111
9.13 GEMFREQ .................................................................................................... 112
  9.13.1 Unique Options ....................................................................................... 112
9.14 GEMLIST ........................................................................................................ 113
9.15 IPSYN ........................................................................................................... 113
9.16 KEYMAP ....................................................................................................... 114
  9.16.1 Sample Runs ............................................................................................ 114
13.4  CHAT2CA ................................................................. 213
13.5  CHAT2CONNL .......................................................... 213
13.6  CHAT2ELAN ............................................................ 214
13.7  CHAT2PRAAT .......................................................... 214
13.8  CHAT2XMAR ............................................................ 214
13.9  CHSTRING .............................................................. 214
13.10 CMDI ........................................................................ 216
13.11 COMBINE ................................................................. 216
13.12 COMPOUND .................................................................. 216
13.13 COMBTIER ............................................................... 217
13.14 CONNL2CHAT .......................................................... 217
13.15 CP2UTF ...................................................................... 217
13.16 DATACLEAN ............................................................. 217
13.17 DATES ........................................................................ 218
13.18 DELIM ........................................................................ 218
13.19 ELAN2CHAT ............................................................. 218
13.20 FIXBULLETS ............................................................... 219
13.21 FIXIT ........................................................................... 219
13.22 FIXLANG ................................................................. 219
13.23 FIXMP3S ................................................................. 219
13.24 FLO ........................................................................... 219
13.25 INDENT ....................................................................... 219
13.26 INSERT ........................................................................ 220
13.27 JOINITEMS ................................................................. 220
13.28 LAB2CHAT ............................................................... 220
13.29 LIPP2CHAT .............................................................. 220
13.30 LONGTIER .................................................................... 220
13.31 LOWCASE ................................................................. 220
13.32 MAKEMOD ................................................................. 221
13.33 OLAC ......................................................................... 222
13.34 ORT ........................................................................... 222
13.35 PRAAT2CHAT ........................................................... 222
13.36 QUOTES ...................................................................... 222
13.37 REPEAT ....................................................................... 223
13.38 RETRACE ................................................................. 223
13.39 RTFIN ......................................................................... 223
13.40 SALTIN ........................................................................ 223
13.41 SILENCE ..................................................................... 223
13.42 SPREADSHEET ............................................................ 223
13.43 SUBTITLES ................................................................. 224
13.44 SYNCODING ............................................................... 224
13.45 TEXTIN ................................................................. 224
13.46 TIERORDER .............................................................. 225
13.47 TRIM ........................................................................... 225
13.48 TRNFIX ................................................................. 225
13.49 UNIQ ........................................................................... 225
13.50  USEDLEX ......................................................................................... 225
14  References ................................................................................................. 226
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 Child Language Data Exchange System (CHILDES) Project, as well as other database systems, such as AphasiaBank and CABank, that form a part of the overall TalkBank system. For a full description of CHAT, please download and read the CHAT manual from the childes.talkbank.org webpage. You can also download data from that page and read about the various corpora in the Database Manuals.

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 of 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.
1. Chapter 1 explains how to install and configure CLAN. This process has different steps, depending on whether you are using Windows or Mac OS X.
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 – Mac OS X

Here is how to install and configure CLAN for Mac OS X:

1. If you need to permit downloading of non-AppStore apps, go to SystemPreferences / Security / General, open the lock, and click on "Anywhere".
2. Next, point your browser at http://childes.talkbank.org/clan and download the Mac version of CLAN. Click to open up clan.dmg and then click to start the installer. It will install in your Applications folder and your working directory will be: Applications/CLAN/work.
3. Drag the CLAN file icon into the dock to create a link for easy access.
4. You may also want to create a link to Applications/CLAN/ in your “favorites” list.
5. Go to System Preferences and select Keyboard. Check the two boxes there in order to use standard function keys and to show Character Viewers.
6. You probably already have QuickTime installed. If not, you can get it from here: http://support.apple.com/downloads/#quicktime

1.4 Setting the Download Directory – Mac OS X

While working with this tutorial, it is a good idea to configure your browser to go to your working directory, which should be Applications/CLAN/work. For current versions of browsers, here are the pathways to locate this setting:

- Firefox 30.x: Firefox/Preferences/General/Save files to/Choose (find /work)
- Chrome 35.x: Chrome/Prefrences/Show advanced/Downloads/Change (find /work)
- Safari 7.x: Safari/Preferences/Save downloaded files to:/Other (find /work)
- Internet Explorer: Gearbox/ViewDownloads/Options/Browse (find /work)

If you are using Safari, you can also check the box at the bottom of the General tab that says Open "safe" files after downloading to make the process easier.

1.5 Installing CLAN – Windows

Here is how to install and configure CLAN for Windows:

1. Point your browser at http://childes.talkbank.org/clan and download the Windows version of CLAN. (Current versions of CLAN are no longer compatible with Windows 95/98/ME.)
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. You may already have QuickTime installed. If not, you can get it from http://support.apple.com/downloads/#quicktime.
1.6 Setting the Download Directory – Windows

While working with this tutorial, it is a good idea to configure your browser to go to your working directory, which should be c:/TalkBank/CLAN/work. For current versions of browsers, here are the pathways to locate this setting:

• Internet Explorer: Gearbox/ViewDownloads/Options/Browse (find /work)
• Firefox 30.x: Firefox/Preferences/General/Save files to/Choose (find /work)
• Chrome 35.x: Chrome/Preferences/Show advanced/Downloads/Change (find /work)
2 Using the Web

In this chapter, we will first survey some of the community resources available at the CHILDES and 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 CHILDES homepage at http://childes.talkbank.org, take a 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. **Overviews.** Introductory materials in English, Chinese, French, Spanish, and Japanese.

Next, take a look at the links under Contact
1. **Membership Lists:** the username is talk and the password is talk.
2. **Joining Membership Lists.
3. **How to subscribe to Mailing Lists** – hosted by googlegroups.com

Now, go to http://talkbank.org and take a quick look at the homepages for:
1. AphasiaBank
2. BilingBank
3. SLABank
4. PhonBank
5. CABank
6. Second Language Resources

2.2 Downloading Materials

Now that you have your download directory configured, you should download additional material for this tutorial from http://childes.talkbank.org.

- **this tutorial** http://childes.talkbank.org/clan/tutorial.zip
- **MOR grammar** http://childes.talkbank.org/morgrams/eng.zip
- **CLAN manual** http://childes.talkbank.org/manuals/CLAN.pdf
- **CHAT manual** http://childes.talkbank.org/manuals/CHAT.pdf
- **Brown transcripts** http://childes.talkbank.org/data/Eng-NA-MOR/brown.zip
- **Database manuals** http://childes.talkbank.org/manuals/
  http://talkbank.org/manuals/

Rather than printing out the long manuals, it is best to keep them in your /work folder and access them through Adobe Reader. In the CHAT and CLAN manuals, you can double-click on items in the Table of Contents and you will go directly to the linked section.
On Windows, to unpack zipped files like brown.zip, do not use the "Extract All" option; instead double-click on the .zip file and then drag the folder to the “work” shortcut on your Desktop.

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. If you are using Firefox, go to Tools/Add-Ons/Plugins and choose to always activate the QuickTime plug-in. You do not have to do this for other browsers.
2. Click on the Browsable Database link on the CHILDES homepage. When the new page opens, glance over the instructions. You can always come back to read these in detail later.
3. In the left column, click on Eng-UK / Forrester / biggirl.cha.
4. 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.
5. 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.
6. If you have a good broadband connection and get garbled audio, click on the ResetQuickTime application that is provided in the /CLAN/lib folder to fix the problem.
7. 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 particular segments to play.
8. You can also run some basic CLAN programs through the Command Line interface on the left. For example, in /DementiaBank/Pitt/Dementia on talkbank.org, you can use: freq +s"*|ProbableADl" +t@ID *.cha +u or you could look for “MCI” or “Vascular”.

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. Click on Database, then Transcripts and then /Eng-UK/Forrester.zip
2. If you have set your downloads directory to /work, the file will go there.
3. If it is not already unzipped, you should unzip it.
4. Downloading video is a slow process, so choose the exact file you wish to study. In this case, we will choose biggirl.cha. Put it in the same directory as the transcript file.
5. In the Database section of the homepage, click on Transcripts and then Eng-UK/Forrester. Finally click to download biggirl.mov.
   a. For Firefox, right-click and choose Save Link As
b. For Safari and Chrome, right-click and choose Download Linked File

c. For Internet Explorer, right-click and choose Save Target As
3 Basic Commands 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 up and you can type commands into this window. If the window does not open automatically, then type Control-d (Windows) or ⌘-d (Macintosh).

[Image of Commands window]

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.

Setting the Working Directory

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

1. Press the working button in the Command window (see screen grab above).
2. Locate the directory that contains the desired files. For this tutorial, please select the examples directory inside the CLAN directory as your working directory.
3. Press the **Select Current Directory** button (see next screen image).

After selecting the current 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.

After you have set the working directory, go through the same process to set the **lib** directory. You do not need to worry about setting the **output** directory. By default, it is the same as the working directory. Also, for now, you do not need to worry about the **mor lib**.

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

**The Recall Button**

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

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

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

**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.

**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. It should look like the screenshot on the next page. Scroll down to the file **sample.cha** and double-click on it to move it to the right. The files on the right will be the ones used for analysis. The **Remove** button that appears under the **Files for Analysis** scrolling list is used to eliminate files from the selected data set. The **Clear** button removes all the files you have added. The 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”.

TIERS Button

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

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

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

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

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

**Wildcards**

A wildcard uses the asterisk symbol (*) to take the place of something else. For example, if you want to run this command across a group of ten files all ending with the extension .cha, you can enter the command in this form:
Wildcards can be used to refer to a group of files (*.cha), a group of speakers (CH*), or a group of words with a common form (*ing). To see how these could work together, try out this command:

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

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

**Output Files**

When you run the command

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

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

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

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

**Redirection**

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

\[
\text{kwal +sbunny -w2 +w2 0042.cha}
\]

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 original transcript will open up 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 Runs

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

\[
\text{freq +t*CHI 0042.cha}
\]

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

*CHI: the dog [/] dog barked.

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

### 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 particular combination of words with COMBO, it is necessary to precede the combination with +s (e.g., +s"kitty^kitty") in the command line. The symbol ^ specifies that the word kitty is immediately followed by the word kitty. The output of the command used above is as follows:

```
> combo +tMOT +s"kitty^kitty" 0042.cha
kitty^kitty
combo +tMOT +skitty^kitty 0042.cha
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 according to 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.
4 Advanced Commands

This chapter provides a series of CLAN commands designed to illustrate a fuller range of options available in some of the most popular CLAN programs. With a few exceptions, the commands are designed to run on the Adler directory included in the CLAN tutorial distribution. 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 each command from this document and paste it into the CLAN Commands window and then hit a carriage return.

```
kwal +t*PAR +s"slipper*" *.cha
run KWAL on the Participant looking for "slipper" in all the files

kwal +t*PAR +s"because" adler23a.cha
run KWAL on the Participant looking for "because" in adler23a.cha

kwal +t*PAR +s"(be)cause" adler23a.cha +r2
run KWAL on the Participant looking specifically for "because" transcribed as (be)cause (produced as "cause") in adler 23a.cha

kwal +t*PAR +s@whwords.cut adler23a.cha
run KWAL on the Participant on the list of words in the whwords.cut in adler23a.cha
(The whwords.cut file is automatically found in the /lib/pos folder)

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

combo +t*PAR +sfairy^godmother +u *.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 +s"inf[^v]*" +t*PAR +t%mor adler01a.cha
run COMBO on the Participant's %mor tier to find all combinations of infinitive and verb in adler01a.cha

mortable +leng +t*PAR *.cha
run MORTABLE using the "eng.cut" script file on the Participant to get a spreadsheet summary with percentages of parts of speech and bound morphemes in all the files

mortable +leng +t*PAR +o4 *.cha
same as above with output in raw numbers instead of %
```
maxwd +g2 +t*PAR *.cha
run MAXWD on the Participant to get the longest utterance in words in all files

eval +t*PAR +u *.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

timedur +t*PAR +d10 *.cha
run TIMEDUR on the Participant, creating a spreadsheet with ratio of words and utterances over time duration for all files

gem +t*PAR +t%mor +sSandwich +n +d1 adler07a.cha
run GEM on the Participant, including the %mor line, using the “Sandwich” gem with lazy gem marking, outputting legal CHAT format for adler07a.cha

vocd +t*PAR +d3 +r6 *.cha
run VOCD on the Participant, output to spreadsheet only, and exclude repetitions and revisions in all the files

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

same thing, but excluding printing of the results for the self-repetitions
The next commands all use the FREQ program to illustrate various options:

freq +t*PAR +o adler01a.cha
run FREQ on the Participant tier and get output in order of descending frequency for adler01a.cha

freq +t*PAR +d2 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 *.cha
same, on all the files in Adler

freq +t@"ID=*|Anomic|*" +d2 *.cha
same, but only include Anomics

freq +t*PAR +d3 adler01a.cha
run FREQ on the Participant tier and get type token ratio only in a spreadsheet for adler01a.cha

freq +t%mor +t*PAR -t* +o 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*PAR +t%mor -t* +s"@r*-"o-"%" +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 +s@
learn about how to use the powerful +s@ switch

freq +t*PAR +s+&uh +s+&um +o adler01a.cha
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"xxx" adler01a.cha
run FREQ on the Participant tier and count instances of unintelligible jargon for adler01a.cha

freq +t*PAR +s"xxx" +d 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&=ges* adler01a.cha
run FREQ on the Participant tier, counting instances of gestures for adler01a.cha
freq +t*PAR +r6 -s"<\* n:uk*>" +o 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 +s"@r-*.o-%" +d6 adler01a.cha
run FREQ on the Participant tier, counting stems, and displaying the associated part of speech and bound morpheme info for adler01a.cha

freq +t*PAR +s"@l-*.o-%" +o adler01a.cha
run FREQ on the Participant tier for part of speech info and getting output in order of descending frequency for adler01a.cha

freq +t*PAR +s"[: Cinderella]" +d6 +u *.cha
run FREQ on the Participant tier, counting all error productions associated with “Cinderella” as the target word, showing error production and error code merged across all files

freq +t*PAR +s@r-happy* +u *.cha
run FREQ on the Participant tier, counting instances and parts of speech for "happy" as a stem, merging across all files

freq +t*PAR +s"[\* *]*" +d6 adler17a.cha
run FREQ on the Participant tier, counting all word errors, and showing target replacements and error codes for adler17a.cha

freq +t*PAR +s"[\* s*]*" *.cha
run FREQ on the Participant tier, counting all semantic errors in all files
add +d6 for the actual errors and targets
add +d for the error in its full context

freq +t*PAR +s"<+ *>" adler01a.cha
run FREQ on the Participant tier, counting all post-codes for adler01a.cha

freq +t*PAR +s"<+ cir>" +u *.cha
run FREQ on the Participant tier, counting all utterances coded as circumlocutions, and merging output across all files
add +d for actual utterances

freq +t*PAR +s"@l-det#" +o adler01a.cha
run FREQ on the Participant, counting words coded on %mor tier as determiners and det:num, and getting output in order of descending frequency for adler01a.cha
add +fdeterminer if you want a file created from this output
freq +t*PAR +s"@l-v,l-cop,l-aux:*j-mod,l-mod:*j-part" adler01a.cha
run FREQ on the Participant, counting words coded on %mor tier as verbs, copulas, auxiliaries and modals for adler01a.cha

freq +t*PAR +s"@r-*l-v,l-cop,l-aux:*j-mod,l-mod:*j-part,o-%" adler01a.cha
run FREQ on the Participant, counting stems of words coded on the %mor tier as verbs, copulas, auxiliaries and modals for adler01a.cha

freq +t*PAR +s"@l-pro*" adler01a.cha
run FREQ on the Participant, counting words coded on %mor tier as pronouns (all types) for adler01a.cha

freq +t*PAR +s"@r-*l-pro*,o-%" adler01a.cha
run FREQ on the Participant, counting stems of words coded on %mor tier as pronouns (all types) for adler01a.cha

freq +t*PAR +s"@l-n,l-n:*" +o adler01a.cha
run FREQ on the Participant, counting words coded on %mor tier as nouns (all types), and get output in order of descending frequency for adler01a.cha

freq +t*PAR +s"@r-*l-n,l-n:*o-%" +o adler01a.cha
run FREQ on the Participant, counting stems of words coded on %mor tier as nouns (all types), and getting output in order of descending frequency for adler01a.cha

freq +t*PAR +s@0list.cut +c3 +u *.cha
run FREQ on the Participant, searching for a list of words in a cut file 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 +d *.cha
same with +d added for outputting the original utterance

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
5 Exercises

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

5.1 Contrasting Four Measures

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

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

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

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

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

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

5.2 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 /childes/clan/lib/ne20. The command string we used to compute MLU for all five children is:

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

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

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

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

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

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

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

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

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.

5.3 **MLU5 Analysis**

The second CLAN analysis we will perform computes the mean length in morphemes of each child’s five longest utterances. To do this, we will run MAXWD 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 68.cha
```

+gl Identify the longest utterances in terms of morphemes
+c5 Identify the five longest utterances
+d1 Output the data in CHAT format
68.cha The child language transcript to be analyzed

The output of this command is:

```
*CHI: <I want to see the other box> [?] .
%mor: pro:sub|I v|want inf|to v|see det|the qn|other n|box .
*CHI: there's a dolly in there [= box] .
%mor: pro:exist|there~cop|be&3S det|a n|doll-DIM prep|in pro:dem|there .
*CHI: that's [= book] the “morning (.) noon and night” .
%mor: pro:dem|that~cop|be&3S det|the bq|bq n|morn|noon coord|and n|night eq|eq .
*CHI: it's [= contents of box] crayons and paper .
%mor: pro|it-aux|be&3S n|crayon-PL coord|and n|paper .
*CHI: pop g(o)es the weasel .
%mor: n|pop v|go-3S det|the n|weasel .
```

We can analyze these results directly for MLU by adding the “piping” symbol which sends the above results from MAXWD directly to MLU:

```
maxwd +t*CHI +g1 +c5 +d1 68.cha | mlu
```

The results of this full command are then:

```
From pipe input
MLU for Speaker: *CHI:
```
MLU (xxx, yyy and www are EXCLUDED from the utterance and morpheme counts):
   Number of: utterances = 5, morphemes = 32
   Ratio of morphemes over utterances = 6.400
   Standard deviation = 0.800

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

```
maxwd +t*CHI +g1 +c5 +d1 14.cha | mlu > 14.ml5.cex
maxwd +t*CHI +g1 +c5 +d1 55.cha | mlu > 55.ml5.cex
maxwd +t*CHI +g1 +c5 +d1 66.cha | mlu > 66.ml5.cex
maxwd +t*CHI +g1 +c5 +d1 68.cha | mlu > 68.ml5.cex
maxwd +t*CHI +g1 +c5 +d1 98.cha | mlu > 98.ml5.cex
```

To run all five commands in sequence automatically, we put the batch file in our working directory with a name such as batchml5.cex and then enter the command
```
bash batch ml5
```
This command will produce five output files.

### 5.4 MLT Analysis

The third analysis we will perform is to compute MLT (Mean Length of Turn) for both child and mother. Note that, unlike the MLU program, the CLAN program MLT includes the symbols xxx and yyy in all counts. Thus, utterances that consist of only unintelligible vocal material still constitute turns, as do nonverbal turns 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: \( \frac{2.184}{5.991} = 0.365 \).

### 5.5 TTR Analysis

The fourth CLAN analysis we will perform for each child is to compute the TTR or type–token ratio. For this we will use the FREQ command. By default, FREQ ignores the strings xxx (unintelligible speech) and www (irrelevant speech researcher chose not to transcribe). It also ignores words beginning with the symbols 0, &, +, - , or #. Here we were interested not in whether the child uses plurals or past tenses, but how many different vocabulary items she uses. Therefore, we wanted to count cats and cat as two tokens (i.e., instances) of the word-type cat. Similarly, we wanted to count play and played as two tokens under the word-type play. To make these distinctions correctly, we need to use MOR and POST to create a %mor line for our transcript. The process of doing this is described in Chapter 11. For now, we will assume that the transcripts already have this %mor line. In that case, the command we use is:

```
freq +t*CHI +s"@r-*,o-%" +f *.cha
```

The only new element in this command is `+s"@r-*,o-%"`. The `+s` option tells FREQ to search for and count certain strings. The `r-*` part of this switch tells FREQ to look only at the roots or lemmas that follow the | symbol in the %mor line. The `o-%` part of the switch tells FREQ to ignore the rest of the material on the %mor line. The output generated from this analysis goes into five files. For the 68.cha input file, the output is 68.frq.cex. At the end of this file, we find this summary analysis:

```
83  Total number of different item types used
244  Total number of items (tokens)
0.340  Type/Token ratio
```

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

### 5.6 Generating Language Profiles

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

**Table 5: New England 20 Means**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Mean</th>
<th>SD</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of different item types used</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total number of items (tokens)</td>
<td>244</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type/Token ratio</td>
<td>0.340</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The distribution of MLU50 scores was quite skewed, with the majority of children who produced at least 50 utterances falling in the MLU range of 1.00-1.30. As noted earlier, 17 of the 48 children failed to produce even 50 utterances. At this age the majority of children in the sample are essentially still at the one-word stage, producing few utterances of more than one word or morpheme. Like MLU50, the shape of the distributions for 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:

\[
\frac{\text{child's score} - \text{group mean}}{\text{standard deviation}}
\]

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

### Table 6: Z-scores for Five Children

<table>
<thead>
<tr>
<th>Child</th>
<th>MLU50</th>
<th>MLU5</th>
<th>TTR</th>
<th>MLT Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>0.26</td>
<td>0.21</td>
<td>1.65</td>
<td>-0.16</td>
</tr>
<tr>
<td>55</td>
<td>-0.30</td>
<td>-0.15</td>
<td>-0.36</td>
<td>-0.53</td>
</tr>
<tr>
<td>66</td>
<td>-0.16</td>
<td>-0.11</td>
<td>-0.64</td>
<td>-0.84</td>
</tr>
<tr>
<td>68</td>
<td>2.30</td>
<td>2.72</td>
<td>-0.86</td>
<td>1.98</td>
</tr>
<tr>
<td>98</td>
<td>-0.96</td>
<td>-0.74</td>
<td>-0.63</td>
<td>-0.08</td>
</tr>
</tbody>
</table>

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

It is not the case, however, that all five of our target children have flat profiles. Some children show marked strengths or weaknesses relative to their peers in particular domains. For example, Subject 14, although very close to the mean in terms of utterance length (MLU50 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 with the exception of 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. In order to get a more detailed picture of a child’s language production, we might choose to include other indices, or to further refine the measures we use. For example, we might compute TTR based on a particular number of words, or we might time-sample by examining the number of word types and word tokens the child produced in a given number of minutes of mother–child interaction. We might also consider other measures of conversational competence, such as number of child initiations and responses; fluency measures, such as number of retraces or hesitations; or pragmatic measures, such as variety of speech acts produced. Computation of some of these measures would require that codes be entered into the transcript prior to analysis; however, the CLAN analyses themselves would, for the most part, simply be variations on the techniques 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.

5.7 Further Exercises

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

1. Compute the length in morphemes of each target child’s single longest utterance at 20 months. Compare with the MLU of the five longest utterances. Consider why a researcher might want to use MLU of the five longest rather than MLU of the single longest utterance.

2. Use the +z option to compute TTR on each child’s first 50 words at 32 months. Then do the same for each successive 50-word band up to 300. Check the output each time to be sure that 50 words were in fact found. If you specify a range of 50 words where there are fewer than 50 words available in the file, FREQ still performs the analysis, but the output will show the actual number of tokens found. What do you observe about the stability of TTR across different samples of 50 words?

3. Use the MLU and FREQ programs to examine the mother’s (*MOT) language to her child at 20 months and at 32 months. What do you observe about the length/complexity and lexical diversity of the mother’s speech to her child? Do they remain generally the same across time or change as the child’s language develops? If you observe change, how can it be characterized?

4. Perform the same analyses for the four target children for whom data are available at age 32 months. Use the data given earlier to compute z-scores for each target child on each measure (MLU 50 utterances, MLU of five longest utterances, TTR, MLT ratio). Then plot profiles for each of the target children at 32 months. What consistencies and inconsistencies do you see from 20 to 32
months? Which children, if any, have similar profiles at both ages? Which children's profiles change markedly from 20 to 32 months?

5. Conduct a case study of a child you know to explore whether type of activity and/or interlocutor affect mean length of turn (MLT). Videotape the child and mother engaged in two different activities (e.g., bookreading, having a snack together, playing with a favorite toy). On another occasion, videotape the child engaged in the same activities with an unfamiliar adult. If it is not possible to videotape, you may audiotape and supplement with contextual notes. Transcribe the interactions in CHAT format. You may wish to put each activity in a separate file (or see CLAN manual for how to use the program GEM). Compare the MLT ratio for each activity and adult–child pair. Describe any differences you observe.
6 The Editor

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

![New File Window]

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.

6.1 The Interactive Tutorial

There are two ways to learn the use of the editor. One way is to work through the current chapter and the next one on media linkage. A probably more effective way is to download the interactive tutorial at \url{http://childes.talkbank.org/clan/tutorial.zip} and work through that tutorial using the two chapters here as backup reference. The interactive tutorial was first composed by Spencer Hazel at Roskilde University, Denmark and then modified to include additional features. The tutorial consists of a number of small files that you work through in a specified order, beginning with 01install.rtf. It then reviews how to get materials from the web, and how to run the commands in Chapter 2. Then it shows you how to transcribe into CHAT files, link to video, align overlaps, insert graphics, send audio to Praat, create gesture codes, check reliability, and other features. The material on linkage overlaps with what is given in Chapter 4 below and the material
on gesture analysis overlaps with Chapter 6. This manual will not make further reference to the interactive tutorial. However, it is highly recommended as an excellent introduction to the practice of transcription.

6.2 **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 have to uncheck the CHAT Mode button on the **Mode** pulldown menu. In CHAT Mode, the editor facilitates the typing of new CHAT files and the editing of existing CHAT files. If your file has the extension .cha, you will automatically be placed into CHAT Mode when you open it. To indicate that you are in CHAT Mode, the bar at the bottom of the editor window displays [E][CHAT].

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

You should stay in CHAT mode until you have learned the basic editing commands. You can insert characters by typing in the usual way. Movement of the cursor with the mouse and arrow keys works the same way 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.

6.3 **File, Edit, and Font Menus**

The functions of opening files, 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.

6.4 **Default Window Positioning and Font Control**

Often you may find that you want to control the way in which the CLAN editor displays windows. There are two basic things you may want to control. First, you may wish to control the position of the window on the screen when you first open it. In order to control the default window position, you need to open an empty new file using `⌘-n`. Then resize and reposition the window to the form and position that you want and close it. After this, both old and new files that you open will appear in this same position. The
system for controlling the default Font uses a slightly different method. To set a default font, open an old file, select the whole text using control-A, and then select the font you want under the Font menu. Whenever you select a new font, it becomes your current default font. The recent fonts you have used appear at the top of CLAN’s Font menu. If you want to select a new default, go through this same process again.

6.5 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 http://talkbank.org/CABank/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.

6.6 Setting Special Colors

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.

6.7 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 similar to the Find dialog. However, this dialog allows you to find a particular 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. You then are led through the words in this replacement file one by one. The form of that file is like this:

\begin{verbatim}
   String_A
   Replacement_A
   String_B
   Replacement_B
   ...
\end{verbatim}

6.8 Hiding Tiers

To hide or unhide a certain dependent tier, type Esc-4. (Remember to always release the escape key before typing the next key). Then you type e to exclude a tier and %mor for the morphological tier. If you want to exclude all tiers, you type just %. To reset the tiers and to see them all, you type Esc-4 and then r.

6.9 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://www.fon.hum.uva.nl/praat) and place your cursor in front of a bullet for a sound segment. Selecting Send to Sound Analyzer then sends that clip to the Praat window for further analysis. To run Praat in the background without a GUI, you can also send this command from a Perl or Tcl script:

\begin{verbatim}
   system ("\"C:\\Program Files\\Praatcon.exe\" myPraatScript.txt"
\end{verbatim}

6.10 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.

6.11 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.
6.12 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 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.
6.13 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] bary.cha 1
```

The [E] entry indicates that you are in editor mode and the [chat] entry indicates that you are in CHAT Mode. In order to begin coding, you first want to set your cursor on the first utterance you want to code. 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 bary.cha file in the samples directory to test this out. 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 your library directory and select one of the demo codes files beginning with the word “code.” We will use codes1.cut for our example.

6.13.1 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 either the cursor keys, the plus and minus keys or a mouse click. If a code is selected, it will be highlighted. You can enter it by hitting the carriage return or double-clicking it. Next, we see the second level of the coding scheme.

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

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

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

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

If you receive the message “Finish coding current tier” in response to a command (as, for example, when trying to change to editor mode), use *Esc-c* to extricate yourself from the coding process. At that point, you can reissue your original command. Here is a summary of the commands for controlling the coding window. On Macintosh, use the command key instead of the control key. Remember to release the esc (escape) key before the next character.

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>esc-c</td>
<td>finish current code</td>
</tr>
<tr>
<td>esc-c (again)</td>
<td>finish current tier</td>
</tr>
<tr>
<td>control-z</td>
<td>undo</td>
</tr>
<tr>
<td>control-t or F1</td>
<td>finish current tier and go to next</td>
</tr>
<tr>
<td>esc-t</td>
<td>restrict coding to a particular speaker</td>
</tr>
<tr>
<td>esc-esc</td>
<td>go to the next speaker</td>
</tr>
<tr>
<td>esc-s</td>
<td>show subcodes under cursor</td>
</tr>
</tbody>
</table>

### 6.13.2 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 /coder folder in the /lib folder included with the CLAN program distribution. In the next paragraphs, we will explain the construction of the codes-basic.cut file in that folder. The first line of your codes-basic.cut file is:

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

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

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

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

```
\ +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 particular level is copied across the branches so that all of the siblings on a given level have the same set of offspring. A more extensive example of a file that uses this type of inheritance is the system for error coding given in the codeserr.cut file in the /lib/coder folder distributed with CLAN.

If not all codes at a given level occur within each of the codes at the next highest level, each individual combination must be spelled out explicitly and the +s option should not be used. The second line in the file should declare the name for your dependent tier. It should end with a tab, so that the tab is inserted automatically in the line you are constructing. A single codes.cut file can include coding systems for many different dependent tiers with each system in order in the file and beginning with an identifier such as $spa:. Setting up the codes.cut file properly is the trickiest part of Coder Mode. Once properly specified, however, it rarely requires modification. If you have problems getting the editor to work, chances are the problem is with your codes.cut file.
7 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. Each transcript should be associated with one and only one media file. To keep your project well organized it is best if the media file name matches the transcript file name. The second field in the @Media header tells whether the media is audio, video, or missing.

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

```
*ROS: alert [!] alert ! \textcolor{red}{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.

To use any of these methods, you need to have a digitized audio or video file. Audio
files can be in either .wav or .mp3 format. Video files can be in any video format that can be played by QuickTime. You will also need to have QuickTime installed on your machine. To practice these methods, you can download the CLAN tutorial from http://childes.talkbank.org/clan/tutorial.zip. In that tutorial, you want to work with the material in the folder labeled as “transcribing”. That folder includes skeleton files that allow you to work through most of the steps discussed in the following sections.

7.1 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. As long as 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. **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-l (insert time code).

5. **Changing the waveform window.** The +H and -H buttons on the left allow you to increase or decrease the amount of time displayed in the window. The +V and -V buttons allow you to control the amplitude of the waveform. The *L and *R buttons can be used to enable or disable the left and right channels in a stereo recording. When the asterisk is present, the channel is enabled. When it is removed, it is disabled.

6. **Scrolling.** At the bottom of the sound window is a scroll-bar that allows you to
move forward or backward in the sound file (please note that scrolling in the sound file can take some time as the sound files for long recordings are very large and take up processing capacity).

7. **Utterance -> Waveform Display.** In order to highlight the section of the waveform associated with a particular utterance, you need to triple-click on the bullet following the utterance you want to replay. You must triple-click at a point just before the bullet to get reliable movement of the waveform. If you do this correctly, the waveform will redisplay. Then you can replay it by using command-click.

8. **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.

9. **Changing the current segment.** To modify the extent of the current highlighted segment in the waveform window, you can use these four functions:
   - 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.
   - 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.
   - 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.
   - 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.
   - N.B. If you, place your cursor right next to a bullet, 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.

10. **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.

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 word “CLAN”. If you click on this line, you will see three additional numbers. The first is the beginning and end time of the current window in seconds. The second is the 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.
7.2 **Transcriber Mode**

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

### 7.2.1 Linking to an already existing transcript

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

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

### 7.2.2 To create a new transcript

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

1. Open a blank file.
2. Make sure that your media file is in your working directory along with your new blank file.
3. Enter these basic headers:
   @Begin
   @Languages: eng (language codes are in section 5.2 of the CHAT manual)
   @Media: filename, audio

   Here, “filename” is the name of your sound file. Keep is simple without spaces. Do not include the extension. If you are using video, then write “video” instead of “audio”.

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, Ctrl-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 right hand corner. You can also expand the bullets {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}".

7.2.3 Sparse Annotation

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

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

To learn to do video linking, it is best to work with the tutorial materials available from http://childes.talkbank.org/clan/tutorial.zip. The tutorial uses a conversation between a clinician and a person with aphasia describing a picture story. The final version you are trying to create is 09final.cha in the /transcribing folder.

1. You may want to open this at the beginning. After the file opens, you will see some solid black bullets. These are the time markers.
2. If you type Esc-A they will expand and you can see their values. Then type Esc-A again to close them. Remember to always release the escape key before typing the next key.
3. Now place your cursor near the beginning of the file and type Esc-8 for continuous playback. The video window should open and the file should play back. When you want it to stop, double click.
4. To play back a single line, place your mouse just to the left of a bullet, press the command key (Windows: control) and click the mouse. After a few seconds of pause, the segment will play.
5. Try playing different segments, as well as using the continuous movie playback mode, showing and playing from thumbnails, and other video features found in the "Mode" menu.

After exploring the final product, you should go through the files in the /transcribing section in sequence.

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

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

1. 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

7.4 SoundWalker

The SoundWalker 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, for example, the programs plays each step three times before moving on.
3. backspace: The amount of rewinding in milliseconds at the end of each loop.
4. walk pause length: The duration of the pause between loops.
5. playback speed: This setting allows you to speed up or slow down your playback rate.

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

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

You will find all of these options in the "Walker Controller" dialog that you open under the Window menu. Once you open the Walker Controller and thereby enable SoundWalker, the functions described above become enabled.
If you would like to use a real foot pedal with SoundWalker, you can order one (Windows Only) from [www.xkeys.com](http://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.

### 7.5 Export to Partitur Editors

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

### 7.6 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. This function is particularly useful if you choose to hide certain speakers tiers. For example, if you hide the Mother using Esc-4, then when you use Esc-9 it will skip over the segments spoken by the Mother.
7.7 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 up 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 check your work by playing the new alignment using command-click.

The second method uses the four key combinations described in section 7.1 on the Sonic Mode for changing the boundaries of the segment in a sound bullet. 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.

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 have to listen through this untranscribed material during continuous playback. To implement forced skipping, you should open up your bullets using Esc-A. Then go to the end of the bullet after the skip and insert a dash after the second time value. For example, you might change 4035_5230 to 4045_5300. Then close the bullets and save the file.
8 Features

8.1 Shell Commands

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

**accept**  
This command applies only to Macintosh. If you only want to have CLAN look at files that the Macintosh calls TEXT files, then type: accept text. If you want to set this back to all files, type accept all.

**batch**  
You can place a group of commands into a text file which you then execute as a batch. The word batch should be followed by the name of a file in your working directory. Each line of that file is then executed as a CLAN command.

**cd**  
This command allows you to change directories. With two dots, you can move up one directory. If you type a folder’s name and the folder is in the current folder, you can move right to that folder. If you type a folder’s absolute address, you can move to that folder from any other folder. For example, the command cd HardDisk:Applications:CLAN on the Macintosh will take you to the CLAN directory.

**copy**  
If you want to copy files without going back to the Finder, you can use this command. The -q option asks to make sure you want to make the copy.

**del**  
This command allows you to delete files. Using this in combination with the +re switch can be very dangerous. In this combination, the command del * can delete all files from your current working directory and those below it. Please be careful!

**dir**  
This command lists all the files in your current directory.

**info**  
This command displays the available programs and commands.

**list**  
This command lists the files that are currently in your input files list.

**rmdir**  
This command deletes a directory or folder.

**ren**  
This command allows you to change file names in a variety of ways. The rename command can use the asterisk as a wildcard for files in which there is a period. You can change case by using -u for upper and -l for lower. You can change extensions by using wildcards in file names. The -c and -t switches allow you to change the creator signature and file types recognized.
by Macintosh. Usually, you will want to have TEXT file types. CLAN produces these by default and you should seldom need to use the -t option. You will find that the -c option is more useful. On the Macintosh, if you want a set of files to have the icon and ownership for CLAN, you should use this command:

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

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

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

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

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

type

This command displays a file in the CLAN output window.

### 8.2 Online Help

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

### 8.3 Commands 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.

### 8.4 Macros

CLAN can store up to 10 Macros. Typing `Esc-n` will open up 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.

### 8.5 Testing CLAN

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

8.6 Bug Reports

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

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

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

8.7 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.
# 9 Analysis Commands Table

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

CLAN also includes two other major groups of commands. The first group is used to perform morphosyntactic analysis on files by tagging words for their part of speech and detecting grammatical relations. These programs are discussed in Chapter 11. In addition, CLAN includes a large group of Utility commands that are described in Chapter 13.

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAINS</td>
<td>60</td>
<td>Tracks sequences of interactional codes across speakers.</td>
</tr>
<tr>
<td>CHECK</td>
<td>64</td>
<td>Verifies the correct use of CHAT format.</td>
</tr>
<tr>
<td>CHIP</td>
<td>67</td>
<td>Examines parent-child repetition and expansion.</td>
</tr>
<tr>
<td>COMBO</td>
<td>73</td>
<td>Searches for complex string patterns.</td>
</tr>
<tr>
<td>COOCUR</td>
<td>81</td>
<td>Examines patterns of co-occurrence between words.</td>
</tr>
<tr>
<td>DIST</td>
<td>82</td>
<td>Examines patterns of separation between speech act codes.</td>
</tr>
<tr>
<td>DSS</td>
<td>83</td>
<td>Computes the Developmental Sentence Score.</td>
</tr>
<tr>
<td>EVAL</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>FREQ</td>
<td>93</td>
<td>Computes the frequencies of the words in a file or files.</td>
</tr>
<tr>
<td>FREQMERG</td>
<td>108</td>
<td>Combines the outputs of various runs of FREQ.</td>
</tr>
<tr>
<td>FREQPOS</td>
<td>109</td>
<td>Tracks the frequencies in various utterance positions.</td>
</tr>
<tr>
<td>GEM</td>
<td>110</td>
<td>Finds areas of text that were marked with GEM markers.</td>
</tr>
<tr>
<td>GEMFREQ</td>
<td>112</td>
<td>Computes frequencies for words inside GEM markers.</td>
</tr>
<tr>
<td>GEMLIST</td>
<td>113</td>
<td>Lists the pattern of GEM markers in a file or files.</td>
</tr>
<tr>
<td>IPSYN</td>
<td>113</td>
<td>Computes IPSYN scores</td>
</tr>
<tr>
<td>KEYMAP</td>
<td>114</td>
<td>Lists the frequencies of codes that follow a target code.</td>
</tr>
<tr>
<td>KIDEVAL</td>
<td>115</td>
<td>Computes a wide variety of indices using other programs.</td>
</tr>
<tr>
<td>KWAL</td>
<td>116</td>
<td>Searches for word patterns and prints the line.</td>
</tr>
<tr>
<td>MAXWD</td>
<td>120</td>
<td>Finds the longest words in a file.</td>
</tr>
<tr>
<td>MLT</td>
<td>121</td>
<td>Computes the mean length of turn.</td>
</tr>
<tr>
<td>MLU</td>
<td>124</td>
<td>Computes the mean length of utterance.</td>
</tr>
<tr>
<td>MODREP</td>
<td>129</td>
<td>Matches the child’s phonology to the parental model.</td>
</tr>
<tr>
<td>MORTABLE</td>
<td>132</td>
<td>Create a frequency table of parts of speech and affixes.</td>
</tr>
<tr>
<td>PHONFREQ</td>
<td>132</td>
<td>Computes the frequency of phonemes.</td>
</tr>
<tr>
<td>RELY</td>
<td>134</td>
<td>Measures reliability across two transcriptions.</td>
</tr>
<tr>
<td>SCRIPT</td>
<td>136</td>
<td>Compares transcripts to target scripts</td>
</tr>
<tr>
<td>TIMEDUR</td>
<td>136</td>
<td>Computes time durations of utterances and pauses</td>
</tr>
<tr>
<td>VOCD</td>
<td>138</td>
<td>Computes the VOCD lexical diversity measure.</td>
</tr>
</tbody>
</table>
9.1 CHAINS

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

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

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

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

9.1.1 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
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
```

```
Speakers *MOT:

# chains  $nia:fp  $nia:gi  $nia:pa  $npp:pa  $npp:yq
0 1 0 0 0 3
2 0 0 0 0 2
0 0 0 1 0 10
0 0 1 0 0 13
```

```
Speakers *CHI:

# chains  $nia:fp  $nia:gi  $nia:pa  $npp:pa  $npp:yq
0 1 0 0 0 1
2 0 0 0 0 1
0 0 0 0 0 1
0 0 1 0 0 1
0 1 1 1 1 1
```

```
SP Part.  0 0 0 0 0
SP/Total 0 0 0 0 0
```

```
Speakers *CHI:

# chains  $nia:fp  $nia:gi  $nia:pa  $npp:pa  $npp:yq
0 1 0 0 0 1
2 0 0 0 0 1
0 0 0 0 0 1
0 0 1 0 0 1
0 1 1 1 1 1
```
You can use CHAINS to track two coding tiers at a time. For example, one can look at chains across both the \%cod and the \%sit tiers by using the following command. This command also illustrates the use of the +c switch, which allows the user to define units of analysis lower than the utterance. In the example file, the [c] symbol is used to delimit clauses. The following command makes use of this marking:

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

The output from this analysis is:

<table>
<thead>
<tr>
<th>Speaker markers: 1=*MOT, 2=*CHI</th>
<th>$a</th>
<th>$b</th>
<th>$c</th>
<th>$d</th>
<th>line #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$ext</td>
<td>$why</td>
<td>2 $ext</td>
<td>$why</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 $mor</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALL speakers:</th>
<th>$a</th>
<th>$b</th>
<th>$c</th>
<th>$d</th>
</tr>
</thead>
<tbody>
<tr>
<td># chains</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Avg leng</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Std dev</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Min leng</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Max leng</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9.1.2 Unique Options

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

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

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

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

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

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

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

```
chains +d +t%spa chains.cha +s$nia:%
```
**+wN** Sets the width between columns to \( N \) characters.

### 9.2 **CHECK**

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

#### 9.2.1 How CHECK Works

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

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

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

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

#### 9.2.2 CHECK in CA Mode

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

1. Each utterance should begin with a number and a speaker code in the form `#:speaker:<whitespace>`.
2. There should be paired parentheses around pause numbers.
3. Numbers marking pause duration are allowed on their own line.
4. Latching should be paired.
5. The double parentheses marking comments should be paired.
6. Overlap markers should be paired.
7. Superscript zeros should be paired.
8. The up-arrow, down-arrow, and zeros are allowed inside words.

9.2.3 Running CHECK

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

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

9.2.4 Restrictions on Word Forms

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

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

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

CHECK also prohibits dashes within words in many contexts. Dashes were eliminated from most contexts in 1998, but are still available for certain special cases. If there is no an '@' symbol before the dash then it is legal if the word starts with a capital letter, otherwise it is always illegal.

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

9.2.5 Some Hints

1. Use CHECK early and often, particularly when you are learning to code in CHAT. When you begin transcribing, check your file inside the editor using Esc-L, even before it is complete. When CHECK complains about something, you can learn right away how to fix it before continuing with the same error.

2. If you are being overwhelmed by CHECK errors, you can use the +d1 switch to limit error reports to one of each type. Or you can focus your work first on eliminating main line errors by using the -t% switch.

3. Learn how to use the query-replace function in your text editor to make general changes and CHSTRING to make changes across sets of files.

9.2.6 Unique Options

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

```
check +e
```

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

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

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

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

+g3 Without the +g3 switch, CHECK does minimal checking for the correctness of the internal contents of words. With this switch turned on, the program makes sure that words do not contain numbers, capital letters, or spurious apostrophes.
CHECK also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

9.3 **CHIP**

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

9.3.1 **The Tier Creation System**

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

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

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

```
@Begin
@Participants: MOT Mother, CHI Child
*MOT: what’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.
```
The output also includes a long set of summary statistics which are discussed later. In the first part of this output, CHIP has introduced four different dependent tiers:

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

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

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

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

### 9.3.2 The CHIP Coding System

The CHIP coding system includes aspects of several earlier systems (Bohannon & Stanowicz, 1988; Demetras, Post, & Snow, 1986; Hirsch-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
$SUB$ 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.

### 9.3.3 Word Class Analysis

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

\[
\text{chip} +c\text{CHI} +b\text{MOT} +g +\text{articles.cut} \text{ chip.cha}
\]

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

\begin{verbatim}
*CHI: a hat!
*MOT: yes that’s the hat.
*adu: $\text{EXA:that} \; \text{EXA:hat} \; \text{ADD:yes} \; \text{SUB:the} \; \text{MADD:'s} \; \text{DIST} = 1
$\text{REP} = 0.50
\end{verbatim}

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 $\%\text{adu}$ 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.

### 9.3.4 Summary Measures

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

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total # of Utterances</td>
<td>The number of utterances for all speakers regardless of the number of intervening utterances and speaker identification.</td>
</tr>
<tr>
<td>Total Responses</td>
<td>The total number of responses for each speaker type regardless of amount of overlap.</td>
</tr>
<tr>
<td>Overlap</td>
<td>The number of responses in which there is an overlap of at least one word stem in the source and response utterances.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>No Overlap</td>
<td>The number of responses in which there is NO overlap between the source and response utterances.</td>
</tr>
<tr>
<td>Avg_Dist</td>
<td>The sum of the DIST values divided by the total number of overlapping utterances.</td>
</tr>
<tr>
<td>%_Overlap</td>
<td>The percentage of overlapping responses over the total number of responses.</td>
</tr>
<tr>
<td>Rep_Index</td>
<td>Average proportion of repetition between the source and response utterance across all the overlapping responses in the data.</td>
</tr>
<tr>
<td>*_OPS</td>
<td>The total (absolute) number of add, delete, exact, or substitution operations for all overlapping utterance pairs in the data.</td>
</tr>
<tr>
<td>%_*_OPS</td>
<td>The numerator in these percentages is the operator being tracked and the denominator is the sum of all four operator types.</td>
</tr>
<tr>
<td>*_WORD</td>
<td>The total (absolute) number of add, delete, exact, or substitution words for all overlapping utterance pairs in the data.</td>
</tr>
<tr>
<td>%_*_WORDS</td>
<td>The numerator in these percentages is the word operator being tracked and the denominator is the sum of all four word operator types.</td>
</tr>
<tr>
<td>MORPH_*</td>
<td>The total number of morphological changes on exactly matching stems.</td>
</tr>
<tr>
<td>%<em>MORPH</em>*</td>
<td>The total number of morphological changes divided by the number of exactly matching stems.</td>
</tr>
<tr>
<td>AV_WORD_*</td>
<td>The average number of words per operation across all the overlapping utterance pairs in the data.</td>
</tr>
<tr>
<td>FRONTED</td>
<td>The number of lexical items from the word list that have been fronted.</td>
</tr>
<tr>
<td>IMITAT</td>
<td>?? Number of imitations.</td>
</tr>
<tr>
<td>%_IMITAT</td>
<td>?? The numerator is the number of imitations and the denominator is ???.</td>
</tr>
</tbody>
</table>
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.

9.3.5 Unique Options

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

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

+\textbf{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.

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

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

Standard CLAN wildcards may be used anywhere in the word list. When the transcript uses CHAT-style morphological coding (e.g., I’ve), only words from the word list file will match to stems in the transcript. In other words, specific morphology may not be traced within a word list analysis. Note that all of the operation and word-based summary statistics are tabulated with respect to the word list only. The word list option may be used for any research purpose including grammatical word classes, number terms, color terms, or mental verbs. Note also that the -\textit{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.

### 9.4 COMBO

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

#### 9.4.1 Composing Search Strings

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

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Type</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediately FOLLOWED by</td>
<td>Boolean</td>
<td>^</td>
</tr>
<tr>
<td>inclusive OR</td>
<td>Boolean</td>
<td>+</td>
</tr>
<tr>
<td>logical NOT</td>
<td>Boolean</td>
<td>!</td>
</tr>
<tr>
<td>repeated character</td>
<td>metacharacter</td>
<td>*</td>
</tr>
<tr>
<td>single character</td>
<td>metacharacter</td>
<td>_</td>
</tr>
<tr>
<td>quoting</td>
<td>metacharacter</td>
<td>\</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

9.4.2 Examples of Search Strings

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

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

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

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

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

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

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

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

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

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

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

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

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

```
combo +s"the^!*^!(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.

### 9.4.3 Referring to Files in Search Strings

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

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

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

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

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

### 9.4.4 Cross-tier Combo

Particular dependent tiers can be included or excluded by using the +t option immediately followed by the tier code. By default, COMBO excludes the header and dependent code tiers from the search and output. However, when the +t switch is used for 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 in order 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. The same effect could also be obtained by using the piping feature.

When two tiers are in one-to-one correspondence, you can create similar analyses more easily by using the +d7 switch with FREQ or the +s@ switch when comparing the `%mor` line with the main line.

### 9.4.5 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 programs work on “clusters” instead of lines. The notion of a cluster is particularly important for search programs, such as COMBO and KWAL. A cluster can be defined as a single utterance by a single speaker, along with all of its dependent tiers.
By default, CLAN programs work on a single cluster at a time. For COMBO, one can extend this search scope to a pair of contiguous clusters 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
```

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

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

**9.4.6 Tracking Final Words**

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

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

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

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

**9.4.7 Tracking Initial Words**

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

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

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

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

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

**9.4.8 Limiting with COMBO**

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

```
combo +t*MOT +t%spa sample.cha +s'want^to' +d | freq
```

The +t*MOT switch 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. The +d option tells the program to output the matching clusters from sample.cha without any non-CHAT identification information. Then the results are sent through a “pipe” indicated by the | symbol to the FREQ program, which conducts an analysis on the main line. The results could also be piped on to other programs such as MLU or KEYMAP or they can be stored in files.

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

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

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

### 9.4.9 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.

### 9.4.10 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

The use of the +d4 switch was described in the previous section.

#### +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 `airplane 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 particular 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. In order to do this
you would use a command of the following shape:

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

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

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

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

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

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

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.

### 9.5 COOCUR

The COOCCUR program tabulates co-occurrences 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.

```plaintext
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-occurrences of codes on the %mor line can be searched using commands such as this example:

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

### 9.5.1 Unique Options

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

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

- **+s** Select either a word or a file of words with @filename to search for.
COOCCUR also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

have a final delimiter. It can also be used to insert final periods on other lines.

9.6 \textit{DIST}

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

\texttt{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.

9.6.1 Unique Options

\textbf{+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:

\texttt{dist +b: sample.cha}

\textbf{+d} Output data in a form suitable for statistical analysis.

\textbf{+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.

\textbf{+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.

\textit{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.

9.7 **DSS**

This program is designed to provide an automatic computation of the Developmental Sentence Score (DSS) of Lee and Canter (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.

9.7.1 **CHAT File Format Requirements**

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

1. All utterances must have delimiters, and imperatives must end with an exclamation mark.
2. Incomplete or interrupted utterances must end either with the +... or the +/. codes.
3. Only the pronoun “I” and the first letter of proper nouns should be in uppercase.
4. Utterances that contain a noun and a verb in a subject-predicate relation in an unusual word order must contain a [+ dss] postcode after the utterance delimiter in order 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.

9.7.2 **Selection of a 50-sentence Corpus**

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

1. **The corpus should contain 50 complete sentences.** A sentence is considered complete if it has a noun and a verb in the subject-predicate relationship. To check for this, the program looks for a nominal phrase followed by a verb. Imperatives such as “Look!” also are included. Imperative sentences must have end with an exclamation mark. Immature sentences containing word order reversals such as “car a garage come out” or “hit a finger hammer Daddy” also should be included. However, these sentences must contain the [+ dss] code after the utterance delimiter on the main tier to be included in the analysis.

2. **The speech sample must be a block of consecutive sentences.** To be representative, the sentences constituting the corpora must occur consecutively in a block, ignoring incomplete utterances. The analyst may use his or her
discretion as to which block of sentences are the most representative. The DSS program automatically includes the first 50 consecutive sentences in the transcript. If you wish to start the analysis at some other point, you can use the +z switch in combination with KWAL and piping to DSS.

3. **All sentences in the language sample must be different.** Only unique child sentences will be included in the corpora. DSS automatically analyzes each sentence and excludes any repeated sentences.

4. **Unintelligible sentences should be excluded from the corpus.** The DSS program automatically excludes any sentences containing unintelligible segments. Thus, any sentence containing 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.

### 9.7.3 Automatic Calculation of DSS

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

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

### 9.7.4 Sentence Points

DSS assigns a sentence point to each sentence if it “meets all the adult standard rules” (Lee, p. 137). Sentence points should be 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.
9.7.5 DSS Output

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

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

<table>
<thead>
<tr>
<th>Sentence</th>
<th>IP</th>
<th>PP</th>
<th>MV</th>
<th>SV</th>
<th>NG</th>
<th>CNJ</th>
<th>IR</th>
<th>WHQ</th>
<th>S</th>
<th>TOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>I like this.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>I like that.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>I want hot dog.</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>I like it.</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>what this say.</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Developmental Sentence Score: 4.2

9.7.6 DSS Summary

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

Indefinite Pronouns (IP) (A)
1 it, this, that
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
uncontracted negatives with have: I have not eaten it.
any other pro-aux + neg forms: you’re not, he’s not
any other aux-negative contractions: aren’t, couldn’t

Conjunction (CNJ) (F)
and
but
so, and so, so that
or, if
where, when, how, while, whether, (or not), till, until, unless, since,
before, after, for, as, as + adjective + as, as if, like, that, than
wh-words + infinitive: I know how to do it.

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

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

9.7.7 DSS for Japanese
DSS can be used for Japanese data to provide an automatic computation of the Developmental Sentence Score for Japanese (DSSJ; Miyata, & al., 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 +drulesjp.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 morphological codes refer to the codes used in JMOR06 and WAKACHI2002 v.5.

Verb Final Inflection (Vlast)
PAST (tabeta), PRES (tabera), IMP:te (tabete!)
HORT (tabeyoo), CONN (tabete...)
COND:tara (tabetara)
CONN&wa (tabecha), GER (tabe), NEG&IMP:de (tabenaida!)
5 IMP (tabeโร), NEG&OBL (tabenakucha)

Verb Middle Inflection (Vmid)
1 COMPL (tabechai), NEG (tabenai), ASP/subi (tabeteru/tabete iru)
2 DESID (tabetai), POT (taberar eru/taber eru), POL (tabemasu),
   sub|ku (tabete karu), subj (tabete iku)
3 sub|mi (tabete miru), sub|ar (tabete aru), sub|ok (tabete oku),
   sub|age (tabete ageru)
4 PASS (taberareru)
5 sub|mora (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 tabenai), 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.,
Japanese. First Language 33, 2, x-x. http://fia.sagepub.com/content/early/recent
www2.aasa.ac.jp/people/smiyata/papers/Miyata&al_DSSJ_FirstLanguage2013.pdf

9.7.8 How DSS works
DSS relies on a series of rules stated in the rules file, 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: prolit+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 of the rules. Within a rule, DSS checks across conditions in order from top to bottom. Once a match is found, it adds the points for that match and then moves on to the next word. This means that, if a condition assigning fewer points could block the application of a condition assigning more points, you need to order the condition assigning more points before the condition assigning fewer points. 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.


### 9.7.9 Unique Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>+b</strong></td>
<td>Designate which speaker to be analyzed.</td>
</tr>
<tr>
<td><strong>+c</strong></td>
<td>Determine the number of sentences to be included in analysis. The default for this option is 50 sentences. These sentences must contain both a subject and a verb, be intelligible, and be unique and non-imitative. A strict criteria is used in the development of the corpora. Any sentences containing xxx yyy and www codes will be excluded from the corpora.</td>
</tr>
<tr>
<td><strong>+e</strong></td>
<td>Automatically generate a DSS table.</td>
</tr>
<tr>
<td><strong>+s</strong></td>
<td>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&quot;[+ dss]&quot;. To exclude sentences with the [+ imit] postcode, the user should include the following option on the command line: -s&quot;[+ imit]&quot;. These are the only two uses for the +s/-s option.</td>
</tr>
</tbody>
</table>

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

This a program that has been designed primarily for use with the English language files in AphasiaBank. All of these files have been analyzed with MOR and GRASP. However, we hope to extend the program for other languages in the next few years.

9.8.1 Explanation of EVAL Measures

All measures are calculated from the %mor line except for FREQ types and FREQ tokens which are calculated from the speaker tier.

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. Adding +r6 to the EVAL command allows for inclusion of repeated and revised words.

7. FREQ tokens: total word tokens as counted by FREQ. The default does not include repetitions and revisions. Adding +r6 to the EVAL command allows for inclusion of repeated and revised words.

8. FREQ TTR: type/token ratio

9. Verbs/Utt: verbs per utterance: (roughly corresponds to clauses per utterance). Includes verbs, copulas, and auxiliaries followed by past or present participles; does not include modals.

10. % Word Errors: percentage of words that are coded as errors [*]. This item can be displayed as a raw number rather than a percentage by adding +o4 to the EVAL command.

11. 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.

12. mor Words: the number of words used for analysis by MOR. Unlike FREQ, by
default MOR counts pre- and post-clitics as words.

13. Density: measure of propositional idea density. Replicates Turner and Green’s rules (1977) for extracting propositions from text, based on the theory of Kintsch (1974). For a list of the words counted as propositions in each file, add -d to the EVAL command. 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.

The next 17 items are expressed as percentages of total words by default. Adding +o4 to the EVAL command displays them as raw numbers.

14. % Nouns
15. % Plurals
16. % Verbs: includes those tagged by MOR as verb, participle, copula and modal
17. % Aux: auxiliaries
18. % 3S: third person singular
19. % 1S/3S: identical forms for first and third person (e.g. I was, he was)
20. % Past
21. % PastP: past participle
22. % PresP: present participle
23. % prep: prepositions
24. %adv: adverbs
25. %adj: adjectives
26. % conj: conjunctions
27. % det: determiners
28. % pro: pronouns
29. retracing [/\]: number retracings (self-corrections or changes)
30. repetition [\]/: number of repetitions

By default, sentences marked by [+ exc] are excluded from the analysis. To include them, add the +s”[+ exc]” to the EVAL command.

9.8.2 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 programs. Push the Option button, and you will see a list of the files in your working directory folder. Double click on eval_demo.cha and it will be displayed on the right side 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 database options. For this demonstration, you want to compare eval_demo.cha to a sample from the AphasiaBank database, so push the Update Database button. This will download a copy of the AphasiaBank transcript data to the work folder, making it available for comparison with eval_demo.cha.

3. Next you need to make choices about the part of the AphasiaBank database you
want to compare to eval_demo.cha. For this example, 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.

4. 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 from the AphasiaBank protocol. Select Sandwich under Gem choices. Press the OK button, and the Commands window will echo this full command you have constructed:
   \[ \text{eval} @ +t\star\text{PAR}: +d"\text{Anomic}45-65" +g"\text{Sandwich}" +u \]

5. 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 up in Excel. The columns list various outputs in terms of indices and part of speech frequencies.


9.8.3 Unique Options

The following options are specific to the EVAL program. If you use the method described above for pushing the Progs button, you can then use the Eval Options dialog box to construct these same options:

- \( +bS \): add S characters to the morpheme delimiters list (default: -#-+)
- \( -bS \): remove S characters from the morphemes list (-b: empty morphemes list)
- \( +cN \): specify database file \((N == 1: \text{eval} +c1 +t"*par")\)
- \( +dS \): specify database keyword(s) “S” 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

9.9 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.

### 9.9.1 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.

### 9.9.2 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 particular 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. In order to get a listing of the frequencies of such words, you need to put all the words you want to track into a text file, one word on each line by itself, and then use the +s switch with the name of the file preceded by the @ sign, as in this example:

```bash
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
```

### 9.9.3 Searches in Bilingual 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:

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

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

```bash
freq +l +s*@s:spa +d7 *.cha
```
Here is a FREQ count of all Spanish words on speaker tier:

\texttt{freq +l+s@spa *.cha}

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

\texttt{freq +s"[- spa]" *.cha}

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

\texttt{freq +s"<- spa>" *.cha}

Counting only the English tiers:

\texttt{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:

\texttt{+s"[- spa]" +s"[- eng]" +s"<- spa>" +s"<- eng>"}

### 9.9.4 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?

9.9.5 Using Wildcards with FREQ

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

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

The output of this command will be:

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

Note that material after the +s switch is enclosed in 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.

<table>
<thead>
<tr>
<th>String</th>
<th>Output</th>
</tr>
</thead>
</table>
| *-acc  | 1 n:alball-acc  
|        | 1 n:alduck-acc  
|        | 1 n:ilplane-acc |
| *-a__  | 1 n:albaby-all  
|        | 1 n:alball-acc  
|        | 1 n:alduck-acc  
|        | 1 n:ilgrape-all  
|        | 1 n:ilplane-acc |
| N:*|*-all | 1 N:Albaby-all  
|        | 1 N:Ilgrape-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:

<table>
<thead>
<tr>
<th>String</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:A</td>
<td>%</td>
</tr>
<tr>
<td>%-ACC</td>
<td>3 -ACC</td>
</tr>
<tr>
<td>%-A__</td>
<td>2 -ALL</td>
</tr>
<tr>
<td>N:</td>
<td>%l%-ACC</td>
</tr>
<tr>
<td>N:</td>
<td>%l%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>String</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>%!*-ACC</td>
</tr>
<tr>
<td></td>
<td>1 N:duck-acc</td>
</tr>
<tr>
<td></td>
<td>1 N:plane-acc</td>
</tr>
<tr>
<td>N:*</td>
<td>%</td>
</tr>
<tr>
<td></td>
<td>2 N:l</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>String</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$*</td>
<td>frequencies of all the three-level codes in the %spa tier</td>
</tr>
<tr>
<td>$*:%</td>
<td>frequencies of the interchange types</td>
</tr>
<tr>
<td>$%:*:%</td>
<td>frequencies of the speech act codes</td>
</tr>
<tr>
<td>$RES:*: %</td>
<td>frequencies of speech acts within the RES category</td>
</tr>
<tr>
<td>$*:sel: %</td>
<td>frequencies of the interchange types that have SEL</td>
</tr>
</tbody>
</table>
speech acts

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

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

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

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

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

9.9.6 FREQ for the %mor line

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

```
+sm Followed by file name or morpho-syntax
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
@ replacement word preceding [: ...] code marker
* error code inside [* ...] code marker,
and then optionally + or – 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:

```
+sm;*,o%
find all stems and erase all other markers
+sm;*,o%,o~
find all stems and erase all other markers including all postclitics, if present
+sm;*,|adv,o%
```
find all stems of all "adv" and erase all other markers
   +sm;be
find all forms of "be" verb
   +sm;*,|*,o%
find all stems and parts-of-speech and erase other markers
   +sm;*,|*,-,o%
find only stems and parts-of-speech that have suffixes and erase
other markers
   +sm;*,|-,+,o-%
find all stems, parts-of-speech and distinguish those with suffix
and erase other markers
   -sm-* +sm;*,|*,o%
find only stems and parts-of-speech that do not have suffixes and
erase other markers
   +sm|n,|n:*,|n:*,o~
find only noun words with "poss" parts-of-speech postclitic
   +sm|n,|n:* +r4
find all noun words and show postclitics, if they have any
   +sm|n,|n:*,o~
find all noun words and erase postclitics, if they have any

9.9.7 Errors relating to particular morphological codes

FREQ allows you to locate all the errors that correspond to particular 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@delivd@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
```

9.9.8 Lemmatization

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Symbol</th>
<th>Example</th>
<th>Representation</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>#</td>
<td>unwinding</td>
<td>un#v</td>
<td>wind-PRESP</td>
</tr>
<tr>
<td>stem</td>
<td>r</td>
<td>unwinding</td>
<td>un#v</td>
<td>wind-PRESP</td>
</tr>
<tr>
<td>part of speech</td>
<td>l</td>
<td>unwinding</td>
<td>un#v</td>
<td>wind-PRESP</td>
</tr>
<tr>
<td>suffix</td>
<td>-</td>
<td>unwinding</td>
<td>un#v</td>
<td>wind-PRESP</td>
</tr>
<tr>
<td>fusion</td>
<td>&amp;</td>
<td>unwound</td>
<td>un#v</td>
<td>wind&amp;PAST</td>
</tr>
<tr>
<td>translation</td>
<td>=</td>
<td>gato</td>
<td>nlgato=cat</td>
<td>cat</td>
</tr>
<tr>
<td>other</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
To illustrate the use of these symbols, let us look at several possible commands. All of these commands take the form: freq +t%mor -t* filename.cha. However, in addition, they add +s switches as given in the second column. In these commands, the asterisk is used to distinguish across forms in the frequency count and the % sign is used to combine across forms.

<table>
<thead>
<tr>
<th>Function</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>All stems with their parts of speech, merge the rest</td>
<td>+s@&quot;r*,j*,o%&quot;</td>
</tr>
<tr>
<td>Only verbs</td>
<td>+s@&quot;</td>
</tr>
<tr>
<td>All forms of the stem &quot;go&quot;</td>
<td>+s@&quot;r-go&quot;</td>
</tr>
<tr>
<td>The different parts of speech of the stem &quot;go&quot;</td>
<td>+s@&quot;r-go,l*,o%&quot;</td>
</tr>
<tr>
<td>The stem &quot;go&quot; only when it is a verb</td>
<td>+s@&quot;r-go,lv,o%&quot;</td>
</tr>
<tr>
<td>All stems, merge the rest</td>
<td>+s@&quot;r*,o%&quot;</td>
</tr>
</tbody>
</table>

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

### 9.9.9 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:

```bash
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:

```bash
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:

```bash
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:

```bash
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:

```bash
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:

```bash
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

9.9.10 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 particular dependent tiers. By using the +t and -t options, you can choose to include certain dependent tiers and ignore others. For example, if you select a particular main speaker tier, you will be able to choose the dependent tiers of only that particular speaker. Each type of tier has to be specifically 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 tells the program to eliminate data found on the main speaker tier for NIC from the analysis. The +s option tells the program to eliminate all the words that do not match the $INI* string from the analysis. Quotes are needed for this particular +s switch in order to guarantee correct interpretation of the asterisk. In general, it is safest to always use pairs of double quotes with the +s switch. The +z20u option tells the program to look at only the first 20 utterances. Now the FREQ program can perform the desired analysis. This command line will send the output to the screen only. You must use the +f option if you want it sent to a file. By default, the header tiers are excluded from the analysis.

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

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

Then the command to ignore German material would be:

freq -s"<@g>" *.cha

9.9.11 TTR for Lemmas

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

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

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

9.9.12 Studying Unique Words and Shared Words

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

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

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

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

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

freq +y +o +u sample.f*

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

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

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

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

### 9.9.13 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

**+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 +s@|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-PRES
  1 n:gerund|look-PRES
```

**+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 +s@| -cop +sROOT +t\gra +t\mor t.cha

+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.

FREQ also uses several options that are shared with other commands, such as +f, +y 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.

9.9.14 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 command.

If you want to exclude unintelligible words and neologisms, add -s"\-l-neol-l-un" to the command.

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 command and use:

freq +t*PAR +o -s"xx" -s"<\n:uk*>" +d6 *.cha

If you want to send the results to a file instead of having them appear on the computer screen, add +f to the command and the new file will appear in the same folder where the transcript files themselves are located. The new filename appears at the bottom of the
message in the CLAN Output window (in this case, it's kempler02a.frq.cex). You can designate a filename by adding +fwords instead of just +f.

If you want only type/token information, use +d4 in the original command.

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

2. 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.

3. 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"\n* *\n" +t*PAR *.cha

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

If you want to specify which errors you want listed and counted, you can list them as we've done below. 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

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, 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 command line.

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

If you want the context (CHAT transcript line) as well, add +d to the command line instead of +d6.
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"[^ * p: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 located in the folder you put as your working directory.

4. 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 particular type of sentence error, for example jargon, use <+ jar> inside the quotation marks.

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

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

6. 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 +s"@-*,o-\%\" +u *.cha

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

freq +t*PAR +d5 +o +s"r-*,l-v*,l-aux,l-aux:*,.l-part*,l-mod*,o-\%, l-cop*,o-\%\" +u *.cha

If you want file location information and each associated %mor line, use +d instead of +d5.

If you want more information about each verb (e.g., tense) use the +d6 option instead.

If you don't want stems only, remove the o% from the command.

If you want to exclude verbs that were produced in error (coded with any of the [* errorcodes] on the main tier), add -s*\** to command line.

7. 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 +s"@l-n,l-n:*o-%" +u *.cha

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

freq +t*PAR +d5 +o +s"@r-*|l-n,l-n:*o-%" +u *.cha

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 +s"@r-*|l-n,l-n:*o-%" +u *.cha

8. 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 +s"@r-*|l-prep*:o-%" +u *.cha

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

9. 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 +s"@r-*|l-adv*:o-%" +u *gem.cex

9. 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 +s"@r-*|l-adj*:o-%" +u *gem.cex

9.10 FREQMERG

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

freq sample*.cha +f

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

freqmerg *.frq.cex

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

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

9.11 FREQPOS

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

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

freqpos sample.cha

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

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

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

freqpos +d sample.cha

9.11.1 Unique Options

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

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

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

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

9.12 GEM

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

9.12.1 Sample Runs

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

```
gem sample.cha
```

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

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

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

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

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

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

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

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

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

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

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

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

gem +n +sreading

9.12.2 Limiting With GEM

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

gem +sreading +d sample2.cha | freq

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

9.12.3 Unique Options

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

gem +sreading +sbook +g sample2.cha
This will retrieve all of the activities involving reading of books.

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

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

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

9.13 \texttt{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:

\texttt{gemfreq +sarriving sample2.cha}

and you would get the following output:

\begin{verbatim}
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
\end{verbatim}

9.13.1 Unique Options

+\texttt{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.

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

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

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

+\textbf{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.

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

\section*{9.14 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.

\section*{9.15 IPSYN}

The IPSyn Command computes the Index of Productive Syntax (Scarborough, 1990).
Computation of this index requires the presence of an accurate %mor line. Currently the program is implemented only for English. By default, the computation uses the first 100 acceptable utterances in a file. The basic form of the command is:

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

### 9.16 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.

#### 9.16.1 Sample Runs

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

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

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

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

For Unix, this command would have to be changed to 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.

#### 9.16.2 Unique Options

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

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

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

### 9.17 KIDEVAL

KIDEVAL performs analyses similar to 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 particular problems and interests. Given a set of transcripts with %mor lines, KIDEVAL will compute these indices:

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. Verbs/Utt: verbs per utterance. This can be less than 1.0 for young children,
12. TD Words: total number of words for each speaker, as used for TIMEDUR
13. TD Utts: total number of utterances for each speaker,
14. TD Time: total duration in seconds of utterances for each speaker,
15. TD Words/Time: words per second,
16. TD Utts/Time: utterances per second,
17. Word Errors: number of words involved in errors,
18. Utt Errors: number of utterances involved in errors,
19. Retracing [/\]: number of retracings,
20. Repetition [\/] : number of repetitions,
21. DSS Utterances
22. DSS Score
23. VOCD score
24. IPSyn Utterances
25. IPSyn Score
26. The frequencies of each of Brown’s 14 grammatical morphemes.

The default output format is a file with the extension .xls that can be opened directly
with Excel. As in EVAL, sentences marked by [+ exc] are excluded from the analysis. If you want to have them included, you can add the +s”[+ exc]” switch.

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 +s"@|adj @|n" +s"@|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.

### 9.17.1 Unique Options

The following options are specific to the KIDEVAL program.

```
+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
+IF: 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
```

### 9.18 KWAL

The KWAL program outputs utterances that match certain user-specified search words. The program also allows the user to view the context in which any given keyword is used. In order to specify the search words, use the +s option, which allows you to search for either a single word or a whole group of words stored in a file. It is possible to specify as many +s options on the command line as you like.
Like COMBO, the KWAL program works not on lines, but on “clusters.” A cluster is a combination of the main tier and the selected dependent tiers relating to that line. Each cluster is searched independently for the given keyword. The program lists all keywords that are found in a given cluster tier. A simple example of the use of KWAL is:

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

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

### 9.18.1 Tier Selection in KWAL

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

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

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

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

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

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

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

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

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

If you wish to study only material in repetitions, you can use KWAL in this form:
9.18.2 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
   del *.no-s.cex

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
   del *.no-g.cex

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
   del *.no-s.cex

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
   del *.sign_only.cex

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
   del *.sign_only.cex
9.18.3 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
+du: 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. In order to do this you would use a command of the following shape:

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

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

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

9.19 MAXWD

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

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

```
maxwd sample.cha
```

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

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

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

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

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

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

```
maxwd +c7 sample.cha
```
If you want to print out all the utterances above a certain length, you can use this KWAL command:

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

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

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

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

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

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

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

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

### 9.20 MLT

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

#### 9.20.1 MLT Defaults

The exact nature of the MLT calculation depends both on what the program in-
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 excluded, although they can be included using the +a switch.

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

   -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.

9.20.2 Breaking Up Turns

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

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

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

9.20.3 Sample Runs

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

mlt sample.cha
9.20.4 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 the particular participant for the particular file. In order to run this type of analysis you must have an @ID header for each participant you wish to track. Alternatively, you can use the +t switch in the form +t*CHI. In this case, all of the *CHI lines will be examined in the corpus. However, if you have different names for children across different files, you need to use the @ID fields.

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

```
*CHI:  6  6  8 1.333 1.000 1.333
*MOT:  8 743 6.143 1.143 5.375
```

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

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

### 9.21 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 ritualized 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 constituent 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 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 going 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. To isolate 100 sentences, use the +z switch. Brown recommended using 100 utterances. He also suggested that these should be taken from the second page of the transcript. In effect, this means that roughly the first 25 utterances should be skipped. The switch that would achieve this effect in the MLU program is +z25u-125u. This is the form of the command used for MLU-100 in the MEASURES 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 *oh* by default. However, if you want to exclude these filler words, you will need to list them in a file and use the -s switch, as in:

```
mlu -s@excludewords sample.cha
```

You can use +s to include lines that would otherwise be excluded. For example, you may want to use `+s"[+ trn]"` to force inclusion of lines marked with `[+ trn]`. You can 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. In particular, the form xxx@a will appear on the %mor line as wlxxxx and the form xxx@a$n will appear on the %mor line as nxxxx. 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.

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

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

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

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

The program considers the following three symbols to be morpheme delimiters: - # ~ MOR analyses distinguish between these delimiters and the ampersand (&) symbol that indicates fusion. As a result, morphemes that are fused with the stem will not be included in the MLU count. If you want to change this list, you should use the +b option described below. For Brown, compounds and irregular forms were monomorphic. 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.

9.21.1 Including and Excluding in MLU and MLT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### 9.21.2 Unique Options

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

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

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

**+d** You can use this switch, together with the ID specification 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. In order to run this type of analysis, you must have an @ID header for each participant you wish to track. You can use the +t switch in the form +tCHI to examine a whole collection of files. In this case, all of the *CHI lines will be examined in the corpus.

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

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

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

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

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

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

9.22 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 wam mo
    %mod:  aI want mor
    *CHI:  want more bananas.
    %pho:  wa mo nAnA
    %mod:  want mor bAn&nAz
    *CHI:  want more bananas.
    %pho:  wa mo nAnA
    %mod:  want mor bAn&nAz
    *MOT:  you excluded [/ /] excluded [/] xxx yyy www &d do?
    %pho:  yu du
    %mod:  yu du
    @End
You can run the following command on this file to create a model-and-replica analysis for the child’s speech:

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

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

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

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

### 9.22.1 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 +qwww
```

The output of this command will look like this:

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

  is longer than Rep line:
  yu du

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

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

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

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

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

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

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

### 9.22.2 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
```

### 9.22.3 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 combine use of MODREP and COMBO. For example, if you want to find all the words matching accusatives that follow verbs, you first select these utterances by running COMBO with the +d switch and the correct +s switch and then pipe the output to the MODREP command we used earlier.

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

The output of this program is the same as in the previous example. However, in a large input file, the addition of the COMBO filter can make the search much more restrictive and powerful.
9.22.4 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 particular string or file of strings. For example, you can cut down the replica tier output to only those strings ending in “-ing.” If you want to track a series of strings or words, you can put them in a file and use the @filename form for the switch.

+o  This switch limits the shape of the output for the model tier in MODREP to some particular string or file of strings. For example, you can cut down the model tier output to only those strings ending in “-ing” or with accusative suffixes, and so forth. If you want to track a series of strings or words, you can put them in a file and use the @filename form for the switch.

+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.

9.23 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 located in 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 script file. The first line under the AND section at the bottom of the CLAN/lib/mortable/eng.cut file provides 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"
```

### 9.24 PHONFREQ

The PHONFREQ program tabulates all of the segments on the %pho line. For example, using PHONFREQ with no further options on modrep.cha will produce this output:

```
2 A initial = 0, final = 1, other = 1
1 I initial = 0, final = 1, other = 0
3 a initial = 1, final = 1, other = 1
2 m initial = 2, final = 0, other = 0
3 n initial = 1, final = 1, other = 1
2 o initial = 0, final = 2, other = 0
2 w initial = 2, final = 0, other = 0
```

This output tells you that there were two occurrences of the segment /A/, once in final position and once in other or medial position.

If you create a file called alphabet file and place it in your working directory, you can further specify that certain digraphs should be treated as single segments. This is important if you need to look at diphthongs or other digraphs in UNIBET. In the strings in the alphabet file, the asterisk character can be used to indicate any single character. For example, the string *: would indicate any sound followed by a colon. If you have three instances of a:, three of e:, and three of o:, the output will list each of these three separately, rather than summing them together as nine instances of something followed by a colon. Because the asterisk is not used in either UNIBET or PHONASCII, it should never be necessary to specify a search for a literal asterisk in your alphabet file. A sample alphabet file for English is distributed with CLAN. PHONFREQ will warn you that it does not find an alphabet file. You can ignore this warning if you are convinced that you do not need a special alphabet file.

If you want to construct a complete substitution matrix for phonological analysis, you need to add a %mod line in your transcript to indicate the target phonology. Then you can run PHONFREQ twice, first on the %pho line and then on the %mod line. To run on the %mod line, you need to add the +t%mod switch.
If you want to specify a set of digraphs that should be treated as single phonemes or segments, you can put them in a file called alphabet.cut. Each combination should be entered by itself on a single line. PHONFREQ will look for the alphabet file in either the working directory or the library directory. If it finds no alphabet.cut file, each letter will be treated as a single segment. Within the alphabet file, you can also specify trigraphs that should override particular digraphs. In that case, the longer string that should override the shorter string should occur earlier in the alphabet file.

9.24.1 Unique Options

+\textbf{b}  
By default, PHONFREQ analyzes the \texttt{%pho} tier. If you want to analyze another tier, you can use the +\textbf{b} switch to specify the desired tier. Remember that you might still need to use the +\textbf{t} switch along with the +\textbf{b} switch as in this command:

\begin{verbatim}
phonfreq +b* +t*CHI modrep.cha
\end{verbatim}

+\textbf{d}  
If you use this switch, the actual words that were matched will be written to the output. Each occurrence is written out.

+\textbf{t}  
You should use the +\textbf{b} switch to change the identity of the tier being analyzed. The +\textbf{t} switch is used to change the identity of the speaker being analyzed. For example, if you want to analyze the main lines for speaker CHI, you would use this command:

\begin{verbatim}
phonfreq +b* +t*CHI modrep.cha
\end{verbatim}

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

The lexicon could be much smaller if more rules were written to handle derivational morphology. These would handle prefixes such as “non#” and derivational suffixes such as “-al.” The grammar still needs to be fine-tuned in order to catch common overregularizations, although it will never be able to capture all possible morphological errors. Furthermore, attempts to capture over regularizations may introduce bogus analyses of good forms, such as “seed” = “*see-PAST.” Other areas for which more rules need to be written include diminutives, and words like “oh+my+goodness,” which should automatically be treated as communicators.

9.25 \textbf{RELY}

This program has two functions. The first is to check reliability. When you are entering a series of codes into files using the Coder Mode, you will often want to compute the reliability of your coding system by having two or more people code a single file or group of files. To do this, you can give each coder the original file, get them to enter a \texttt{%cod} line and then use the RELY program to spot matches and mismatches. For example, you could copy the sample.cha file to the samplea.cha file and change one code in the line 15. Then enter the command:
rely sample.cha samplea.cha +t%spa

The output in the sample.rly file will look like the basic sample.cha file, but with this additional information for line 15:

%spa:  $RDE:sel:non $RFI:xxx:gone:?"samplea.cha"
      $RFI:xxx:in:?"sample.cha"

By default RELY will examine the main tier and all dependent tiers. 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 can use the +t switch to pick out a line to include. If the command is:

rely +c sample.cha samplea.cha +t%spa

then the program will use sample.cha as the master file for everything except the information on the %spa tier.

The second function of the RELY program is to allow multiple coders to add a series of dependent tiers to a master file. The main lines of master file should remain unchanged as the coders add additional information on dependent tiers. This function is accessed through the +a switch, which tells the program the name of the code from the secondary file that is to be added to the master file, as in

rely +a orig.cha codedfile.cha +t%spa

RELY can also compute Cohen’s Kappa and percentage agreement using the +d switch.

9.25.1 Unique Options

+a   Add tiers from second file to first, master, file.
+ c  Do not check data on nonselected tier.
+d   This command: rely +d rely1.cha rely2.cha will produce these results:

rely1.cha: total utterances: 41
rely2.cha: total utterances: 41
total utterances %: 100.0000

rely1.cha: total words (tokens): 158
rely2.cha: total words (tokens): 160
total words (tokens) matched: 156
total words (tokens) %: 97.5000

To calculate agreement, for example, on the overall category of phonological errors, use:
REL also uses several options that are shared with other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

**9.26 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 the Participant’s Script.

**The Model Script**

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.

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.

**The Participant’s Script**

Transcribe the Participant’s script production in CHAT format and link it to the media. Here are some special transcription considerations for SCRIPT analysis:

- 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. If the Participant skips a line of script, enter 0 for that line (*PAR: 0.).
- If the Participant makes comments (e.g., “I don’t know”, “oh no”) in the midst of an utterance, put those words inside angle brackets with [/] afterward so they will not be counted as extra words in the analysis.
- 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).
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.

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.

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 Error Coding, CHAT Manual, Chapter 14). 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).

Variations

If you want to produce output for all of 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 in 1 .cex file and 1 .xls file 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 (like [: 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

9.27 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:

\texttt{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

9.28 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

\[ \text{type–token Ratio (TTR)} \]. Unfortunately, such measures, including mathematical

transformations of the TTR such as Root TTR, are functions of the number of tokens in

the transcript or language sample — samples containing larger numbers of tokens give

lower values for TTR and vice versa (Richards & Malvern, 1997a). This problem has
distorted research findings (Richards & Malvern, 1997b). Previous attempts to overcome
the problem, for example by standardizing the number of tokens to be analyzed from
each child, have failed to ensure that measures are comparable across researchers who
use different baselines of tokens, and inevitably waste data in reducing analyses to the
size of the smallest sample.

The approach taken in the VOCD program is based on an analysis of the probability
of new vocabulary being introduced into longer and longer samples of speech or writing.
This probability yields a mathematical model of how TTR varies with token size. By
comparing the mathematical model with empirical data in a transcript, VOCD provides a
new measure of vocabulary diversity called D. The measure has three advantages: it is

not a function of the number of words in the sample; it uses all the data available; and it

is more informative, because it represents how the TTR varies over a range of token size.
The measure is based on the TTR versus token curve calculated from data for the
transcript as a whole, rather than a particular TTR value on it.

D has been shown to be superior to previous measures in both avoiding the inherent
flaw in raw TTR with varying sample sizes and in discriminating across a wide range of
language learners and users (Malvern, Richards, Chupere, Purán, 2004; Richards &
Malvern, 1998).

9.28.1 Origin of the Measure

TTRs inevitably decline with increasing sample size. Consequently, any single value
of TTR lacks reliability as it will depend on the length in words of the language sample
used. A graph of TTR against tokens (N) for a transcript will lie in a curve beginning at
the point (1,1) and falling with a negative gradient that becomes progressively less steep
(see Malvern & Richards, 1997a). All language samples will follow this trend, but
transcripts from speakers or writers with high vocabulary diversity will produce curves
that lie above those with low diversity. The fact that TTR falls in a predictable way as the
token size increases provides the basis for our approach to finding a valid and reliable
measure. The method builds on previous theoretical analyses, notably by Brainerd (1982)
and in particular Sichel (1986), which model the TTR versus token curve mathematically
so that the characteristics of the curve for a transcript yields a valid measure of
vocabulary diversity.

Various probabilistic models were developed and investigated in order to arrive at a
model containing only one parameter which increases with increasing diversity and falls
into a range suitable for discriminating among the range of transcripts found in various
language studies. The model chosen is derived from a simplification of Sichel’s (1986)
type–token characteristic curve and is in the form an equation containing the parameter
D. This equation yields a family of curves all of the same general and appropriate shape,
with different values for the parameter D distinguishing different members of this family (see Malvern & Richards, 1997). In the model, D itself is used directly as an index of lexical diversity.

In order to calculate D from a transcript, the VOCD program first plots the empirical TTR versus tokens curve for the speaker. It derives each point on the curve from an average of 100 trials on subsamples of words of the token size for that point. The subsamples are made up of words randomly chosen (without replacement) from throughout the transcript. The program then finds the best fit between the theoretical model and the empirical data by a curve-fitting procedure which adjusts the value of the parameter (D) in the equation until a match is obtained between the actual curve for the transcript and the closest member of the family of curves represented by the mathematical model. This value of the parameter for best fit is the index of lexical diversity. High values of D reflect a high level of lexical diversity and lower diversity produces lower values of D.

The validity of D has been the subject of extensive investigation (Malvern & Richards, 1997; Richards & Malvern, 1997a; Richards & Malvern, 1998; Malvern, Richards, Chupere, Purán, 2004) on samples of child language, children with SLI, children learning French as a foreign language, adult learners of English as a second language, and academic writing. In these validation trials, the empirical TTR versus token curves for a total of 162 transcripts from five corpora covering ages from 24 months to adult, two languages and a variety of settings, all fitted the model. The model produced consistent values for D which, unlike TTR and even Mean Segmental TTR (MSTTR) (see Richards & Malvern, 1997a: pp. 35-38), correlated well with other well validated measures of language. These five corpora also provide useful indications of the scale for D.

9.28.2 Calculation of D

In calculating D, VOCD uses random sampling of tokens in plotting the curve of TTR against increasing token size for the transcript under investigation. Random sampling has two advantages over sequential sampling. Firstly, it matches the assumptions underlying the probabilistic model. Secondly, it avoids the problem of the curve being distorted by the clustering of the same vocabulary items at particular points in the transcript.

In practice each empirical point on the curve is calculated from averaging the TTRs of 100 trials on subsamples consisting of the number of tokens for that point, drawn at random from throughout the transcripts. This default number was found by experimentation and balanced the wish to have as many trials as possible with the desire for the program to run reasonably quickly. The run time has not been reduced at the expense of reliability, however, as it was found that taking 100 trials for each point on the curve produced consistency in the values output for D without unacceptable delays.

Which part of the curve is used to calculate D is crucial. First, in order to have subsamples to average for the final point on the curve, the final value of N (the number of tokens in a subsample) cannot be as large as the transcript itself. Moreover, transcripts
vary hugely in total token count. Second, the equation is an approximation to Sicel’s (1986) model and applies with greater accuracy at lower numbers of tokens. In an extensive set of trials, D has been calculated over different parts of the curve to find a portion for which the approximation held good and averaging worked well. As a result of these trials the default is for the curve to be drawn and fitted for N=35 to N=50 tokens in steps of 1 token. Each of these points is calculated from averaging 100 subsamples, each drawn from the whole of the transcript. Although only a relatively small part of the curve is fitted, it uses all the information available in the transcript. This also has the advantage of calculating D from a standard part of the curve for all transcripts regardless of their total size, further providing for reliable comparisons between subjects and between the work of different researchers.

The procedure depends on finding the best fit between the empirical and theoretically derived curves by the least square difference method. Extensive testing confirmed that the best fit procedure was valid and was reliably finding a unique minimum at the least square difference.

As the points on the curve are averages of random samples, a slightly different value of D is to be expected each time the program is run. Tests showed that with the defaults chosen these differences are relatively small, but consistency was improved by VOCD calculating D three times by default and giving the average value as output.

9.28.3 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.

9.28.4 Preparation of Files

Files should be prepared in correct CHAT format and should pass through CHECK, using the +g3 switch to track down spelling and typographical errors. The FREQ program should then be used to create a complete wordlist that can be scanned for further errors. The output from FREQ also allows the researcher to see exactly what FREQ (and therefore VOCD) will treat as a word type. From this information, an exclude file of non-words can be compiled (e.g. hesitations, laughter, etc). These can then be filtered out of the analysis using the -s switch.

9.28.5 The Output from VOCD

To illustrate the functioning of VOCD, let us use a command that examines the child’s output in the file 68.cha in the NE32 sample directory in the lib folder in the CLAN distribution. The +r6 switch here excludes repetitions, the +s@exclude lists a file of words to be excluded, and the +s”*.-%” instructs CLAN to merge across variations of a base word.
The output of this analysis has four parts:

1. A sequential list of utterances by the speaker selected shows the tokens that will be retained for analysis.
2. A table shows the number of tokens for each point on the curve, average TTR and the standard deviation for each point, and the value of D obtained from the equation for each point. Three such tables appear, one for each time the program takes random samples and carries out the curve-fitting.
3. At the foot of each of the three tables is the average of the Ds obtained from the equation and their standard deviation, the value for D that provided the best fit, and the residuals.
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:

<table>
<thead>
<tr>
<th>tokens</th>
<th>samples</th>
<th>ttr</th>
<th>st.dev</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>100</td>
<td>0.7963</td>
<td>0.067</td>
<td>54.470</td>
</tr>
<tr>
<td>36</td>
<td>100</td>
<td>0.8067</td>
<td>0.054</td>
<td>60.583</td>
</tr>
<tr>
<td>37</td>
<td>100</td>
<td>0.8008</td>
<td>0.059</td>
<td>59.562</td>
</tr>
<tr>
<td>38</td>
<td>100</td>
<td>0.7947</td>
<td>0.056</td>
<td>58.464</td>
</tr>
<tr>
<td>39</td>
<td>100</td>
<td>0.7831</td>
<td>0.065</td>
<td>55.124</td>
</tr>
<tr>
<td>40</td>
<td>100</td>
<td>0.7772</td>
<td>0.054</td>
<td>54.242</td>
</tr>
<tr>
<td>41</td>
<td>100</td>
<td>0.7720</td>
<td>0.064</td>
<td>53.568</td>
</tr>
<tr>
<td>42</td>
<td>100</td>
<td>0.7767</td>
<td>0.057</td>
<td>56.720</td>
</tr>
<tr>
<td>43</td>
<td>100</td>
<td>0.7695</td>
<td>0.051</td>
<td>55.245</td>
</tr>
<tr>
<td>44</td>
<td>100</td>
<td>0.7650</td>
<td>0.057</td>
<td>54.787</td>
</tr>
<tr>
<td>45</td>
<td>100</td>
<td>0.7636</td>
<td>0.053</td>
<td>55.480</td>
</tr>
<tr>
<td>46</td>
<td>100</td>
<td>0.7626</td>
<td>0.057</td>
<td>56.346</td>
</tr>
<tr>
<td>47</td>
<td>100</td>
<td>0.7543</td>
<td>0.052</td>
<td>54.403</td>
</tr>
<tr>
<td>48</td>
<td>100</td>
<td>0.7608</td>
<td>0.050</td>
<td>58.088</td>
</tr>
<tr>
<td>49</td>
<td>100</td>
<td>0.7433</td>
<td>0.058</td>
<td>52.719</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>0.7396</td>
<td>0.049</td>
<td>52.516</td>
</tr>
</tbody>
</table>

D: average = 55.770; std dev. = 2.289
D optimum <55.63; min least sq val = 0.001>

VOCD RESULTS SUMMARY
==================================
Command line:  vocd +t"*CHI" +r6 -s@exclude +s"*-%" 68.cha
File name:  68.cha
Types, Tokens, TTR:  <129, 376, 0.343085>
D optimum values:  <55.36, 55.46, 55.63>
D optimum average:  55.48
9.28.6 Lemma-based Analysis

If you wish to conduct a lemma-based analysis, rather than a word-based analysis, you can do this from the %mor line using a command of this type:

\[ \text{vocd} \ +t*\text{PAR} \ +t%\text{mor} \ -t* \ +t*|\ *+\%%| +s"*|+\%%" \ *.cha \]

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

\[ \text{vocd} \ +t*\text{PAR} \ +t%\text{mor} \ -t* \ +t*|\ *+\%%| +s"*|+\%%" \ -s"\text{neo}|*" \ +f \ *.cha \]

9.28.7 Unique Options

+\text{d} \quad \text{Outputs a list of utterances processed and number of types, tokens and TTR, but does not calculate D.}
+\text{d1}: \quad \text{outputs only VOCD results summary on one line}
+\text{d2}: \quad \text{outputs only type/token information}
+\text{d3}: \quad \text{outputs only type/token information in Excel format}

+\text{b} \quad \text{Various methods for controlling sampling}
+\text{b0}: \quad \text{D_optimum - use split half; even.}
+\text{b1}: \quad \text{D_optimum - use split half; odd.}
+\text{bsN}: \quad \text{D_optimum - size N of starting sample (default 35)}
+\text{blN}: \quad \text{D_optimum - size N of largest sample (default 50)}
+\text{biN}: \quad \text{D_optimum - size N of increments (default 1)}
+\text{bnN}: \quad \text{D_optimum - the N number of samples (default 100)}
+\text{br}: \quad \text{D_optimum - random sampling with replacement (default: noreplacement)}
+\text{be}: \quad \text{D_optimum - use sequential sampling}

+\text{g} \quad \text{Calls up the limiting relative diversity (LRD) sub-routine to compare the relative diversity of two different word classes coded on the %mor tier. This procedure operates in three stages and extracts the words to be included from the %mor tier where the word classes are coded. First, the speaker, the %mor tier and the file name are specified in the usual way, plus the +g switch to invoke the subroutine:}

\[ \text{vocd} \ +t"*\text{CHI}" \ +t"%\text{mor}" \ -t"*" \ +g \ \text{filename} \]

Second, the user is prompted twice to specify the word classes to be compared. The following would compare verb and noun diversity and limit the analysis to word stems:

\[ +s"v*" \ +t"*\%%" \]
\[ +s"n*" \ +t"*\%%" \]

The first word class entered will be the numerator and the second will be the denominator.
+gnS: compute "limiting type-type ratio" S=NUMERATOR +s directives
-gnS: compute "limiting type-type ratio" S=NUMERATOR -s directives
+gdS: compute "limiting type-type ratio" S=DENOMINATOR +s directives
-gdS: compute "limiting type-type ratio" S=DENOMINATOR -s directives

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

9.29 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:

<table>
<thead>
<tr>
<th>Number of words of each length in characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengths: 1 2 3 4 5 6 7 8 9 Mean</td>
</tr>
<tr>
<td>*CHI: 0 0 0 4 4 0 0 0 0 4.50</td>
</tr>
<tr>
<td>*MOT: 2 4 11 11 11 0 0 0 0 4.00</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of utterances of each of these lengths in words</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengths: 1 2 3 4 5 6 7 8 9 10 Mean</td>
</tr>
<tr>
<td>*CHI: 3 1 1 0 0 0 0 0 0 1.60</td>
</tr>
<tr>
<td>*MOT: 0 1 1 1 2 2 0 0 0 3.77</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of single turns of each of these lengths in utterances</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengths: 1 2 Mean</td>
</tr>
<tr>
<td>*CHI: 0 0 0.00</td>
</tr>
<tr>
<td>*MOT: 0 1 2.00</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of single turns of each of these lengths in words</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengths: 1 2 3 4 5 6 7 8 9 10 11 Mean</td>
</tr>
<tr>
<td>*CHI: 0 0 0 0 0 0 0 0 0 0 0.00</td>
</tr>
<tr>
<td>*MOT: 0 0 0 0 0 0 0 0 0 1 11.00</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of words of each of these morpheme lengths</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengths: 1 2 Mean</td>
</tr>
<tr>
<td>*CHI: 7 1 1.12</td>
</tr>
<tr>
<td>*MOT: 35 7 1.16</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of utterances of each of these lengths in morphemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengths: 1 2 3 4 5 6 7 8 9 10 11 12 Mean</td>
</tr>
<tr>
<td>*CHI: 3 0 2 0 0 0 0 0 0 0 0 0 1.80</td>
</tr>
<tr>
<td>*MOT: 0 0 2 0 1 2 2 0 0 0 0 1 4.46</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>
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.

The 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.
10 Options

This chapter describes the various options or switches that are shared across the CLAN analysis commands. To see a list of options for a given program such as KWAL, type kwal followed by a carriage return in the Commands window. You will see a list of available options in the CLAN Output window.

Each option begins with a + or a -. There is always a space before the + or -. Multiple options can be used and they can occur in any order. For example, the command:

```
kwal +f +t*MOT sample.cha
```

runs a KWAL analysis on sample.cha. The selection of the +f option sends the output from this analysis into a new file called sample.kwa.cex. The +t*MOT option confines the analysis to only the lines spoken by the mother. The +f and +t switches can be placed in either order.

10.1 +F Option

This option allows you to send output to a file rather than to the screen. By default, nearly all of the programs send the results of the analyses directly to the screen. You can, however, request that your results be inserted into a file. This is accomplished by inserting the +f option into the command line. The advantage of sending the program’s results to a file is that you can go over the analysis more carefully, because you have a file to which you can later refer.

The -f switch is used for sending output to the screen. For most programs, -f is the default and you do not need to enter it. You only need to use the -f switch when you want the output to go to the screen for CHSTRING, FLO, and SALTIN. The advantage of sending the analysis to the screen (also called standard output) is that the results are immediate and your directory is less cluttered with nonessential files. This is ideal for quick temporary analysis.

The string specified with the +f option is used to replace the default file name extension assigned to the output file name by each program. For example, the command

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

would create an output file sample.frq.cex. If you want to control the shape of the extension name on the file, you can place up to three letters after the +f switch, as in the command

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

which would create an output file sample.mot.cex. If the string argument is longer than three characters, it will be truncated. For example, the command

```
freq +fmother sample.cha
```

would also create an output file sample.mot.cex.

On the Macintosh, you can use the third option under the File menu to set the directory for your output files. On Windows you can achieve the same effect by using the +f switch with an argument, as in:

```
+fc:
```

This will send the output files to your working directory on c:.
+f".res"
  This sets the extension for your output files.
+f"c:.res"
  This sends the output files to c: and assigns the extension .res.

When you are running a command on several files and use the +f switch, the output will go into several files – one for each of the input files. If what you want is a combined analysis that treats all the input files as one large file, then you should use the +u switch. If you want all of the output to go into a single file for which you provide the name, then use the > character at the end of the command along with an additional file name. The > option can not be combined with +f.

10.2 +J Option

The +j switch can also be used in various programs to track stems, prefixes, suffixes, and so on on the %mor line. The operation of this switch is difficult to understand, so we have created a GUI facility to help the user. After typing “freq”, you then hit the “search” button and it will help you compose your search pattern. You can also get information about this switch by typing

freq +j
which will give you the following information:

Search patterns are composed of three components: head symbols, operators, and string matches. Head symbols come first and they declare the type of form being tracked. There are nine possible head symbols. The first six characters are used for morphological features on the %mor line. The next two (@ and *) refer to codes on the main line. The last one (o) is a filter that removes all non-matched elements from the output.

#  prefix marker
l  part-of-speech marker
;  stem of the word marker
-  suffix marker
&  nonconcatenated morpheme marker
=  English translation for the stem marker
@  replacement word preceding [: ...] code marker
*  error code inside [* ...] code marker
o  all other elements not specified by user

Any of the nine head symbols may then be followed by one of two operators: either a dash (-) or a plus (+). The dash means that the following string must occur. The plus means that the following string is an optional add-on to the basic morphosyntactic form being matched.

After the dash or plus, there must be a final string match that can be identified in one of these three ways:

*  find any match
%  erase any match
string  find "string"

The “o” filter can also be followed by the ~ symbol to erase or ignore all postclitic
elements or by the $ symbol to erase or ignore all preclitic elements.

Search patterns composed in this way can be combined in series, separated by commas. If the head symbols of each search pattern are the same, then they are taken as alternatives or “ors”. If they are different, search patterns define conditions that must all be matched. However, if a plus is used inside a search pattern, then the condition it specifies is optional.

Here are some example commands:

Find all stems and erase all other markers (this finds all lemmas):
\[ +s"@r-*,o-\%" \]

Find all stems of all "adv" and erase all other markers:
\[ +s"@r-*,|adv,o-\%" \]

Find all forms of "be" verb
\[ +s"@r-be" \]

Find all stems and parts-of-speech and erase other markers
\[ +s"@r-*,|-*,o-\%" \]

Find only stems and parts-of-speech that have suffixes and erase other markers
\[ +s"@r-*,|-*,-*,o-\%" \]

Find all stems, parts-of-speech, distinguish suffixes and erase other markers
\[ +s"@r-*,|-*,-*,o-\%" \]

It is also possible to combine both –s and/or +s search patterns and they will have a cumulative effect.

Find only stems and parts-of-speech with no suffixes and erase other markers:
\[ -s"-*" +s"@r*,|*,o-\%" \]

If you combine several +s patterns, they will each provide a separate output:

Find all suffixes and prefixes along with words they were used with:
\[ +s"@#*, +s"@-*, +s"@-\%" \]

List all the distinct prefixes and suffixes:
\[ freq +s"@#*,o\%" +s"@-*,o\%" +s"@-\%,"o\%" \]

Postclitic AND Preclitic exceptions:

Find postclitics with specific Morphosyntactic marker example:
\[ |*,~|* OR ;*,~;* \]

Find preclitics with specific Morphosyntactic marker example:
\[ |*,$|* OR ;*,$;* \]
10.3 +K Option

This option controls case-sensitivity. A case-sensitive program is one that makes a distinction between uppercase and lowercase letters. The CLAN programs, except for FREQ, FREQMERGE, MORTABLE, PHONFREQ, RELY, and TIMEDUR, are case-insensitive by default. 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.

10.4 +L Option

The +l option is used to provide language tags for each and 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.

10.5 +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. As a result, 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.

10.6 +R Option

This option deals with the treatment of material in parentheses.

+r1 Removing Parentheses. Omitted parts of words can be marked by parentheses, as in “(be)cause” with the first syllable omitted. The +r1 option removes the parentheses and leaves the rest of the word as is.

+r2 Leaving Parentheses. This option leaves the word with parentheses.

+r3 Removing Material in Parentheses. This option removes all of the omitted part.

Here is an example of the use of the first three +r options and their resulting outputs, if the input word is “get(s)”: 
Option | Output
---|---
"no option" | gets
"+r1" | gets
"+r2" | get(s)
"+r3" | get

**+r4 Removing Prosodic Symbols in Words.** By default, symbols such as #, /, and : are ignored when they occur inside words. Use this switch if you want to include them in your searches. If you do not use this switch, the strings cat and ca:t will be seen as the same. If you use this switch, they will be seen as different. The use of these prosodic marker symbols is discussed in the CHAT manual.

**+r5 Text Replacement.** By default, material in the form [: text] replaces the material preceding it in the string search programs. 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.

**+r7** Do not remove prosodic symbols (/~^:) in words

**+r8:** Combine %mor tier items with replacement word [: …] and error code [* …] if any from speaker tier.

### 10.7 +S Option

This option allows you to search for a particular string. The +s option allows you to specify the keyword you desire to find. You do this by putting the word in quotes directly after the +s switch, as in +s"dog" to search for the word “dog.” You can also use the +s switch to specify a file containing words to be searched. You do this by putting the file name after +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 particular codes within square brackets, (4) whole utterances associated with particular postcodes, and (5) particular postcodes themselves. Moreover, the switch can be used to include, exclude, or add particular information. The effect of the switch for the five different types across the three functions is described in the following three tables:

Table 4: Search Strings for Inclusion of Five Types of Material

<table>
<thead>
<tr>
<th>Material</th>
<th>Switch</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>+&quot;dog&quot;</td>
<td>find only the word “dog”</td>
</tr>
<tr>
<td>[code]</td>
<td>+&quot;[/]/&quot;</td>
<td>find only this code</td>
</tr>
<tr>
<td>&lt;text&gt;[code]</td>
<td>+&quot;&lt;//&gt;&quot;</td>
<td>find only text marked by this code</td>
</tr>
<tr>
<td>utterance</td>
<td>+&quot;&lt;+imi&gt;&quot;</td>
<td>find only utterances marked with this postcode or precode</td>
</tr>
<tr>
<td>postcode</td>
<td>+&quot;[+imi]&quot;</td>
<td>find only this postcode itself</td>
</tr>
</tbody>
</table>

Table 5: Search Strings for Exclusion of Four Types of Material

<table>
<thead>
<tr>
<th>Material</th>
<th>Switch</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>-&quot;dog&quot;</td>
<td>find all words except the word “dog”</td>
</tr>
<tr>
<td>&lt;text&gt;[code]</td>
<td>-&quot;&lt;//&gt;&quot;</td>
<td>find all text except text marked by this code</td>
</tr>
<tr>
<td>utterance</td>
<td>-&quot;[+imi]&quot;</td>
<td>find all utterances except utterances marked with this postcode</td>
</tr>
</tbody>
</table>

Table 6: Search Strings for Addition of Four Types of Material that are usually excluded by default

<table>
<thead>
<tr>
<th>Material</th>
<th>Switch</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>+&quot;xxx&quot;</td>
<td>add “xxx”</td>
</tr>
<tr>
<td>[code]</td>
<td>+&quot;[/]/&quot;</td>
<td>find all text, plus this code</td>
</tr>
<tr>
<td>utterance</td>
<td>+&quot;&lt;+bch&gt;&quot;</td>
<td>find all utterances, including those marked with the [+ bch] postcode</td>
</tr>
<tr>
<td>postcode</td>
<td>+&quot;[+imi]&quot;</td>
<td>find all text, including this postcode itself</td>
</tr>
</tbody>
</table>

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

\[ \text{freq } +s\text{gf% } +s\text{gf:a } +t\text{cod} \]

and your text has these codes

\[ \text{gf\:a } \text{gf:a } \text{gf:b } \text{gf:c} \]

your output will be

\[ \text{gf} \quad 3 \\
\text{gf:a} \quad 1 \]

Because \text{gf:a} matches specifically to the \text{+s gf:a}, it is excluded from matching \text{+s gf%}.

One can also use the +s switch to remove certain strings from automatic exclusion. For example, the MLU program automatically excludes \text{xxx}, 0, \text{uh}, and words beginning with \& from the MLU count. This can be changed by using this command:

\[ \text{mlu } +s\text{+uh } +s\text{+xxx } +s\text{0} +s\text{&} \text{ file.cha} \]

10.8 +T Option

This option allows you to include or exclude particular tiers. In CHAT formatted files, there exist three tier code types: main speaker tiers (denoted by *), speaker-dependent tiers (denoted by %), and header tiers (denoted by @). The speaker-dependent tiers are attached to speaker tiers. If, for example, you request to analyze the speaker *MOT and all the %cod dependent tiers, the programs will analyze all of the *MOT main tiers and only the %cod dependent tiers associated with that speaker.

The +t option allows you to specify which main speaker tiers, their dependent tiers, and header tiers should be included in the analysis. All other tiers, found in the given file, will be ignored by the program. For example, the command:

\[ \text{freq } +t\text{*CHI } +t\text{spa } +t\text{mor } +t\text{"Group of Mot"} \text{ sample.cha} \]

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:

\[ \text{freq } -t\text{*CHI } -t\text{spa } -t\text{mor } -t\text{"Group of Mot"} \text{ sample.cha} \]

tells FREQ to exclude all the *CHI main speaker tiers together with all their dependent tiers, the %spa and %mor dependent tiers on all other speakers, and all @Situation header tiers from the analysis. All remaining tiers will be included in the analysis.

When the transcriber has decided to use complex combinations of codes for speaker
IDs such as *CHI-MOT for “child addressing mother,” it is possible to use the +t switch with the # symbol as a wildcard, as in these commands:

```
freq +t*CHI-MOT sample.cha
freq +t*#-MOT sample.cha
freq +t*CHI-# sample.cha
```

When tiers are included, the analysis will be done on only those specified tiers. When tiers are excluded, however, the analysis is done on tiers other than those specified. Failure to exclude all unnecessary tiers will cause the programs to produce distorted results. Therefore, it is safer to include tiers in analyses than to exclude them, because it is often difficult to be aware of all the tiers present in any given data file.

If only a tier-type symbol (*, %, @) is specified following the +t/-t options, the programs will include all tiers of that particular symbol type in the analysis. Using the option +t@ is important when using KWAL for limiting (see the description of the KWAL program), because it makes sure that the header information is not lost.

The programs search sequentially, starting from the left of the tier code descriptor, for exactly what the user has specified. This means that a match can occur wherever what has been specified has been found. If you specify *M on the command line after the option, the program will successfully match all speaker tiers that start with *M, such as *MAR, *MIK, *MOT, and so forth. For full clarity, it is best to specify the full tier name after the +t/-t options, including the : character. For example, to ensure that only the *MOT speaker tiers are included in the analysis, use the +t*MOT: notation.

As an alternative to specifying speaker names through letter codes, you can use the form:

```
+tid=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.

As an alternative to specifying speaker names through letter codes, you can use the form:

```
+tid=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.

### 10.9 +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 the result for 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.

10.10 +V Option

This switch gives you the date when the current version of CLAN was compiled.

10.11 +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.

10.12 +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 particular items in above count (Example: +xxxx +xxyy)  
-xS: exclude particular items to exclude from above count

In the MOR and CHIP programs, +x has a different meaning.

10.13 +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 in order 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
```

### 10.14 +Z Option

This option allows the user to select any range of words, utterances, or speaker turns to be analyzed. The range specifications should immediately follow the option. For example:

- `+z10w` analyze the first ten words only.
- `+z10u` analyze the first ten utterances only.
- `+z10t` analyze the first ten speaker turns only.
- `+z10w-20w` analyze 11 words starting with the 10th word.
- `+z10u-20u` analyze 11 utterances starting with the 10th utterance.
- `+z10t-20t` analyze 11 speaker turns starting with the 10th turn.
- `+z10w-` analyze from the tenth word to the end of file.
- `+z10u-` analyze from the tenth utterance to the end of file.
- `+z10t-` analyze from the tenth speaker turn to the end of file.

If the +z option is used together with the +t option to select utterances from a particular speaker, then the counting will be based only on the utterances of that speaker. For example, this command:

```
mlu +z50u +t*CHI 0611.cha
```

will compute the MLU for the first 50 utterances produced by the child. If the +z option is used together with the +s option, the counting will be dependent on the working of the +s option and the results will seldom be as expected. To avoid this problem, you should use piping, as in this example:

```
kwal +d +z1-3u +t*CHI sample.cha | kwal +sMommy
```

If the user has specified more items than exist in the file, the program will analyze only the existing items. If the turn or utterance happens to be empty, because it consists of special symbols or words that have been selected to be excluded, then this utterance or turn is not counted.

The usual reason for selecting a fixed number of utterances is to derive samples that are comparable across sessions or across children. Often researchers have found that samples of 50 utterances provide almost as much information as samples of 100 utterances. Reducing the number of utterances being transcribed is important for clinicians who have been assigned a heavy case load.
You can use the +z switch with KWAL and pipe the results to a second program, rather than using it directly with FREQ or MLU. For example, in order to specifically exclude unintelligible utterances in an MLU analysis of the first 150 utterances from the Target Child, you could use this form:

```
kwal +z150u +d +t*CHI 0042.cha -syyy -sxxx | mlu
```

You can also use postcodes to further control the process of inclusion or exclusion.

### 10.15 Metacharacters for Searching

Metacharacters are special characters used to describe other characters or groups of characters. Certain metacharacters may be used to modify search strings used by the +s/-s switch. However, in order to use metacharacters in the CHSTRING program a special switch must be set. The CLAN metacharacters are:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Any number of characters matched</td>
</tr>
<tr>
<td>%</td>
<td>Any number of characters matched and removed</td>
</tr>
<tr>
<td>%%</td>
<td>As above plus remove previous character</td>
</tr>
<tr>
<td>_</td>
<td>Any single character matched</td>
</tr>
<tr>
<td>\</td>
<td>Quote character</td>
</tr>
</tbody>
</table>

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 that the * symbol does. Unlike the * symbol, however, all the characters matched by the % will be ignored in terms of the way of which the output is generated. In other words, the output will treat “beat” and “bat” as two occurrences of the same string, if the search string is b%t. Unless the % symbol is used with programs that produce a list of words matched by given keywords, the effect of the % symbol will be the same as the effect of the * symbol.

When the percentage symbol is immediately followed by a second percentage symbol, the effect of the metacharacter changes slightly. The result of such a search would be that the % symbol will be removed along with any one character preceding the matched
string. Without adding the additional % character, a punctuation symbol preceding the wildcard string will not be matched and will be ignored. Adding the second % sign can be particularly useful when searching for roots of words only. For example, to produce a word frequency count of the stem “cat,” specify this command:

```plaintext
freq +s"cat-%%" file.cha.
```

The first % sign matches the suffixes and the second one matches the dash mark. Thus, the search string specified by the +s option will match words like: “cat,” “cats,” “cat's,” and “cats'” and FREQ will count all of these words as one word “cat.” If the data file file.cha had consisted of only those four words, the output of the FREQ program would have been: 4 cat. The limitation of this search is that it will not match words like “cats” or “cat's,” because the second percentage symbol is used to match the punctuation mark. The second percentage symbol is also useful for matching hierarchical codes such as $NIA:RP:IN.

The underline character _ is similar to the * character except that it is used to specify any single character in a word. For example, the string b_d will match words like “bad,” “bed,” “bud,” “bid,” and so forth. For detailed examples of the use of the percentage, underline, and asterisk symbols, see the section special characters.

The quote character (\) is used to indicate the quotation of one of the characters being used as metacharacters. Suppose that you wanted to search for the actual symbol (*) in a text. Because the (*) symbol is used to represent any character, it must be quoted by inserting the (\) symbol before the (*) symbol in the search string to represent the actual (*) character, as in “string\*string.” To search for the actual character (\), it must be quoted also. For example, “string\string” will match “string” followed by “\” and then followed by a second “string.”
11 MOR – Morphosyntactic Analysis

The modern study of child language development owes much to the methodological and conceptual advances introduced by Brown (1973). In his study of the language development of Adam, Eve, and Sarah, Roger Brown focused on a variety of core measurement issues, such as acquisition sequence, growth curves, morpheme inventories, productivity analysis, grammar formulation, and sampling methodology. The basic question that Brown was trying to answer was how one could use transcripts of interactions between children and adults to test theoretical claims regarding the child’s learning of grammar. Like many other child language researchers, Brown considered the utterances produced by children to be a remarkably rich data source for testing theoretical claims. At the same time, Brown realized that one needed to specify a highly systematic methodology for collecting and analyzing these spontaneous productions.

Language acquisition theory has advanced in many ways since Brown (1973), but we are still dealing with many of the same basic methodological issues he confronted. Elaborating on Brown’s approach, researchers have formulated increasingly reliable methods for measuring the growth of grammar, or morphosyntax, in the child. These new approaches serve to extend Brown’s vision into the modern world of computers and computational linguistics. New methods for tagging parts of speech and grammatical relations now open up new and more powerful ways of testing hypotheses and models regarding children’s language learning.

Before embarking on our review of computational tools in CHILDES, it is helpful to review briefly the ways in which researchers have come to use transcripts to study morphosyntactic development. When Brown collected his corpora back in the 1960s, the application of generative grammar to language development was in its infancy. However, throughout the 1980s and 1990s (Chomsky & Lasnik, 1993), linguistic theory developed increasingly specific proposals about how the facts of child language learning could illuminate the shape of Universal Grammar. At the same time, learning theorists were developing increasingly powerful methods for extracting linguistic patterns from input data. Some of these new methods relied on distributed neural networks (Rumelhart & McClelland, 1986), but others focused more on the ways in which children can pick up a wide variety of patterns in terms of relative cue validity (MacWhinney, 1987).

These two very different research traditions have each assigned a pivotal role to the acquisition of morphosyntax in illuminating core issues in learning and development. Generativist theories have emphasized issues such as: the role of triggers in the early setting of a parameter for subject omission (Hyams & Wexler, 1993), evidence for advanced early syntactic competence (Wexler, 1998), evidence for early absence functional categories that attach to the IP node (Radford, 1990), the role of optional infinitives in normal and disordered acquisition (Rice, 1997), and the child’s ability to process syntax without any exposure to relevant data (Crain, 1991). Generativists have sometimes been criticized for paying inadequate attention to the empirical patterns of distribution in children’s productions. However, work by researchers, such as
Stromswold (1994), van Kampen (1998), and Meisel (1986), demonstrates the important role that transcript data can play in evaluating alternative generative accounts.

Learning theorists have placed an even greater emphasis on the use of transcripts for understanding morphosyntactic development. Neural network models have shown how cue validities can determine the sequence of acquisition for both morphological (MacWhinney & Leinbach, 1991; MacWhinney, Leinbach, Taraban, & McDonald, 1989; Plunkett & Marchman, 1991) and syntactic (Elman, 1993; Mintz, Newport, & Bever, 2002; Siskind, 1999) development. This work derives further support from a broad movement within linguistics toward a focus on data-driven models (Bybee & Hopper, 2001) for understanding language learning and structure. These accounts formulate accounts that view constructions (Tomasello, 2003) and item-based patterns (MacWhinney, 1975) as the loci for statistical learning.

11.1 Analysis by Transcript Scanning

Although the CHILDES Project has succeeded in many ways, it has not yet provided a complete set of computational linguistic tools for the study of morphosyntactic development. In order to conduct serious corpus-based research on the development of morphosyntax, users will want to supplement corpora with tags that identify the morphological and syntactic status of every morpheme in the corpus. Without these tags, researchers who want to track the development of specific word forms or syntactic structures are forced to work with a methodology that is not much more advanced than that used by Brown in the 1960s. In those days, researchers looking for the occurrence of a particular morphosyntactic structure, such as auxiliary fronting in yes-no questions, would have to simply scan through entire transcripts and mark occurrences in the margins of the paper copy using a red pencil. With the advent of the personal computer in the 1980s, the marks in the margins were replaced by codes entered on a %syn (syntactic structure) or %mor (morphological analysis with parts of speech) coding tier. However, it was still necessary to pour over the full transcripts line by line to locate occurrences of the relevant target forms.

11.2 Analysis by Lexical Tracking

If a researcher is clever, there are ways to convince the computer to help out in this exhausting process of transcript scanning. An easy first step is to download the CLAN programs from the CHILDES website at http://childes.psy.cmu.edu. These programs provide several methods for tracing patterns within and between words. For example, if you are interested in studying the learning of English verb morphology, you can create a file containing all the irregular past tense verbs of English, as listed in the CHILDES manual. After typing all of these words into a file and then naming that file something like irreg.cut, you can use the CLAN program called KWAL with the +s@irreg.cut switch to locate all the occurrences of irregular past tense forms. Or, if you only want a frequency count, you can run FREQ with the same switch to get a frequency count of the various forms and the overall frequency of irregulars. Although this type of search is very helpful, you will also want to be able to search for overregularizations and overmarkings such as “*ranned”, “*runned”, “*goed”, or “*jumpeded”. Unless these are
already specially marked in the transcripts, the only way to locate these forms is to create an even bigger list with all possible overmarkings. This is possible for the common irregular overmarkings, but doing this for all overmarked regulars, such as “*playeded”, is not really possible. Finally, you also want to locate all correctly marked regular verbs. Here, again, making the search list is a difficult matter. You can search for all words ending in –ed, but you will have to cull out from this list forms like “bed”, “moped”, and “sled”. A good illustration of research based on generalized lexical searches of this type can be found in the study of English verb learning by Marcus et al. (1992).

Or, to take another example, suppose you would like to trace the learning of auxiliary fronting in yes-no questions. For this, you would need to create a list of possible English auxiliaries to be included in a file called aux.cut. Using this, you could easily find all sentences with auxiliaries and then write out these sentences to a file for further analysis. However, only a minority of these sentences will involve yes-no questions. Thus, to further sharpen your analysis, you would want to further limit the search to sentences in which the auxiliary begins the utterance. To do this, you would need to dig carefully through the electronic version of the CHILDES manual to find the ways in which to use the COMBO program to compose search strings that include markers for the beginnings and ends of sentences. Also, you may wish to separate out sentences in which the auxiliary is moved to follow a wh-word. Here, again, you can compose a complicated COMBO search string that looks for a list of possible initial interrogative or “wh” words, followed by a list of possible auxiliaries. Although such searches are possible, they tend to be difficult, slow, and prone to error. Clearly, it would be better if the searches could examine not strings of words, but rather strings of morphosyntactic categories. For example, we would be able to trace sentences with initial wh-words followed by auxiliaries by just looking for the pattern of “int + aux”. However, in order to perform such searches, we must first tag our corpora for the relevant morphosyntactic features. The current article explains how this is done.

11.3 Analysis by MOR, POST, and GRASP

So far, our analysis has examined how researchers can use the database through transcript scanning and lexical scanning. However, often these methods are inadequate for addressing broader and more complex issues such as detailed syntactic analysis or the comparisons and evaluations of full generative frameworks. To address these more complex issues, the CHILDES system now provides full support for analysis based on automatic morphosyntactic coding. The core programs used in this work are MOR, POST, and GRASP.

The initial morphological tagging of the CHILDES database relies on the application of the MOR program. Running MOR on a CHAT file is easy. In the simplest case, it involves nothing much more than a one-line command. However, before discussing the mechanics of MOR, let us take a look at what it produces. To give an example of the results of a MOR analysis for English, consider this sentence from eve15.cha in Roger Brown’s corpus for Eve.

*CHI: oop I spilled it.
Here, the main line gives the child’s production and the %mor line gives the part of speech for each word, along with the morphological analysis of affixes, such as the past tense mark (-PAST) on the verb. The %mor lines in these files were not created by hand. To produce them, we ran the MOR program, using the MOR grammar for English, which can be downloaded from http://childes.psy.cmu.edu/morgrams/.

The command for running MOR is nothing more in this case than “mor *.cha”. After running MOR, the file looks like this:

*CHI:  oop I spilled it .
%mor:  int|oop pro|I part|spill-PAST pro|it .

Notice that the word “spilled” is initially ambiguous between the past tense and participle readings. To resolve such ambiguities, we run a program called POST. Running POST for English is also simple. The command is just “post *.cha” After POST has been run, the sentence is then “disambiguated.” Using this disambiguated form, we can then run the GRASP program, which is currently a separate program available from the CHILDES website, to create the representation given in the %xsyn line below:

*CHI:  oop I spilled it .
%mor:  int|oop pro|I v|spill-PAST pro|it .
%xsyn:  1|3|JCT 2|3|SUBJ 3|0|ROOT 4|3|OBJ 5|3|PUNCT

In this %xsyn line, we see that the second word “I” is related to the verb (“spilled”) through the grammatical relation (GR) of Subject. The fourth word “it” is related to the verb through the grammatical relation of Object.

Using GRASP, we have recently inserted dependency grammar tags for all of these grammatical relations in the Eve corpus. In tests run on the Eve corpus, 94% of the tags were assigned accurately (Sagae, Davis, Lavie, MacWhinney, & Wintner, 2007). A further test of GRASP on the Adam corpus also yielded an accuracy level of 94%. For both of these corpora, grammatical relations were mistagged 6% of the time. It is likely that, over time, this level of accuracy will improve, although we would never expect 100% accuracy for any tagging program. In fact, only a few human taggers can achieve 94% accuracy in their first pass tagging of a corpus.

The work of building MOR, POST, and GRASP has been supported by a number of people. Mitzi Morris built MOR in 1997, using design specifications from Roland Hauser. Since 2000, Leonid Spektor has extended MOR in many ways. Christophe Parisse built POST and POSTTRAIN (Parisse & Le Normand, 2000) and continues to maintain and refine them. Kenji Sagae built GRASP as a part of his dissertation work for the Language Technologies Institute at Carnegie Mellon University (Sagae, MacWhinney, & Lavie, 2004a, 2004b). GRASP was then applied in detail to the Eve and Adam corpus by Eric Davis and Shuly Wintner.

These initial experiments with GRASP and computational modelling of grammatical development in the CHILDES corpora underscore the increasing importance of methods from computational linguistics for the analysis of child language data. Together with statistical computational analyses (Edelman, Solan, Horn, & Ruppin, 2004) and neural network analyses (Li, Zhao, & MacWhinney, 2007), we should expect to see increasing
input from computational linguistics, as the morphosyntactic tagging of the CHILDES database becomes increasingly refined and accurate.

The computational design of MOR was guided by Roland Hausser’s (1990) MORPH system and was implemented by Mitzi Morris. The system has been designed to maximize portability across languages, extendability of the lexicon and grammar, and compatibility with the CLAN programs. The basic engine of the parser is language independent. Language-specific information is stored in separate data files. The rules of the language are in data files that can be modified by the user. The lexical entries are also kept in ASCII files and there are several techniques for improving the match of the lexicon to a particular corpus. In order to avoid having too large a lexical file, only stems are stored in the lexicon and inflected forms appropriate for each stem are compiled at run time.

MOR automatically generates a %mor tier in which words are labeled by their syntactic category or “scat”, followed by the pipe separator |, followed by the word itself, broken down into its constituent morphemes.

```
*CHI: the people are making cakes.
%mor: det|the n|people v:aux|be&PRES v|make-ING n|cake-PL.
```

The MOR program looks at each word on the main tier, without regard to context, and provides all possible grammatical categories and morphological analyses, as in the following example with the words “to” and “back.” The caret ^ denotes the multiple possibilities for each word on the main tier.

```
*CHI: I want to go back.
%mor: pro|I v|want inf|to^prep|to v|go adv|back^n|back^v|back.
```

In order to select the correct form for each ambiguous case, the user can either edit the file using Disambiguator Mode or use POST.

One way of restricting the possible categories inserted by MOR is to use the replacement symbol [: text] on the main line for difficult cases. For example, the English form “wanna” could mean either “want to” or “want a”. Similarly, “gotta” could be either “got to” or “got a.” The transcriber can commit to one of these two readings on the main line by using this method:

```
*CHI: I wanna [: want to] go back.
%mor: pro|I v|want inf|to^prep|to v|go adv|back^n|back^v|back.
```

In this example, MOR will only attend to the material in the square brackets and will ignore the form “wanna.”

### 11.4 Exclusions in MOR

Because MOR focuses on the analysis of the target utterance, it excludes a variety of non-
words, retraces, and special symbols. Specifically, MOR excludes:

1. Items that start with &
2. Pauses such as (.)
3. Unknown forms marked as xxx, yyy, or www
4. Data associated with these codes: [/?], [/], [/], and [///].

11.5 Configuring MOR

For MOR to run successfully, you need to configure your grammar files and lexicon files into their proper positions in the MOR library directory. You will want to create a specific library directory for MOR that is distinct from the general CLAN lib directory. It is often convenient to place this MOR library inside the CLAN lib directory. In the MOR library directory, you need these three grammar files on top: ar.cut, cr.cut, and sf.cut. Optionally, you may also want to have a file called dr.cut. Within this directory, you then need to have a subdirectory called lex, which contains all of the various closed and open class lexicon files such as adj.cut, clo.cut, prep.cut, or n.cut. If you have retrieved the MOR grammar from the Internet or the CD-ROM, the materials will already be configured in the correct relative positions. Each separate grammar should be stored in its own folder and you should select the grammar you wish to use by setting the MORLIB location in the commands window.

11.5.1 Grammar and Lexicon Files

MOR relies on three files to specify the morphological processes of the language. They are:

1. The allomorph rules file. This file lists the ways in which morphemes vary in shape. The rules that describe these variations are called “arules.” The name of this file should be ar.cut.
2. The concatenation rules file. This file lists the ways in which morphemes can combine or concatenate. The rules that describe allowable concatenations are called “crules”. The name of this file should be cr.cut.
3. The special form markers file. The CHAT manual presents a series of special form markers that help identify lexical types such as neologisms, familial words, onomatopoeia, or second-language forms. MOR can use these markings to directly insert the corresponding codes for these words onto the %mor line. The sf.cut file includes all of these special form markers. In addition, these types must be listed in the first declaration in the cr.cut file. For English, all this is already done. If you are creating a grammar for another language, you can model your materials on the English example. The syntax of the lines in the sf.cut file is fairly simple. Each line has a special form marker, followed by the category information you wish to see inserted in the %mor line. If you wish to pull out capitalization words as being proper nouns, despite the shape of the special form marker, you can place \c to indicate uppercase before the special form marker. You must then add \l on another line to indicate what you want to have done with lowercase examples. See the English sf.cut file for examples.
In addition to these three grammar files, MOR uses a set of lexicon files to specify the shapes of individual words and affixes. These forms are stored in a group of files in the lexicon folder. The affix.cut file includes the prefixes and suffixes for the language. The other files contain the various open and closed class words of the language. At run time, MOR used the grammar rules to “blow up” the content of the lexicon files into a large binary tree that represents all the possible words of the language.

The first action of the parser program is to load the ar.cut file. Next the program reads in the files in your lexicon folder and uses the rules in ar.cut to build the run-time lexicon. If your lexicon files are fairly big, you will need to make sure that your machine has enough memory. On Macintosh, you can explicitly assign memory to the program. On Windows, you will have to make sure that your machine has lots of memory. Once the run-time lexicon is loaded, the parser then reads in the cr.cut file. Additionally, if the +b option is specified, the dr.cut file is also read in. Once the concatenation rules have been loaded the program is ready to analyze input words. As a user, you do not need to concern yourself about the run-time lexicon. Your main concern is about the entries in the lexicon files. The rules in the ar.cut and cr.cut files are only of concern if you wish to have a set of analyses and labelings that differs from the one given in the chapter of the CHAT manual on morphosyntactic coding, or if you are trying to write a new set of grammars for some language.

11.5.2 Unique Options

**+d** do not run POST command automatically. POST will run automatically after MOR, unless this switch is used or unless the folder name includes the word “train”.

**+eS** Show the result of the operation of the arules on either a stem S or stems in file @S. This output will go into a file called debug.cdc in your library directory. Another way of achieving this is to use the +d option inside “interactive MOR”

**+p** use pinyin lexicon format for Chinese

**+xi** Run MOR in the interactive test mode. You type in one word at a time to the test prompt and MOR provides the analysis on line. This facility makes the following commands available in the CLAN Output window:

```
word - analyze this word
:q  quit - exit program
:c  print out current set of crules
:d  display application of arules.
:l  re-load rules and lexicon files
:h  help - print this message
```

If you type in a word, such as “dog” or “perro,” MOR will try to analyze it and give you its components morphemes. If you change the rules or the lexicon, use :l to reload and retest. The :c and :d switches will send output to a file called debug.cdc in your library directory.
Run MOR in the lexicon building mode. This mode takes a series of .cha files as input and outputs a small lexical file with the extension .ulx with entries for all words not recognized by MOR. This helps in the building of lexicons.

Check lexicon mode, include word location in data files
Check lexicon for ambiguous entries
Check lexicon mode, including capitalized words
Check lexicon for compound words conflicting with plain words
Check lexicon mode, including words with prosodic symbols
Analyze words in lex files

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

When analyzing corpora that contain full utterances in a second or third language marked with [ -*], as in [- spa], it is important to note that MOR excludes these lines by default. They can be included using the +s"[ - *]" switch.

### 11.5.3 MOR Lexicons

Before running MOR on a set of CHAT files, it is important to make sure that MOR will be able to recognize all the words in these files. A first step in this process involves running the CHECK program to make sure that all of the words follow basic CHAT rules, such as not including numbers or capital letters in the middle of words. For more details about these various CHECK requirements, please consult the sections of this manual that describe CHECK.

Once you know that a corpus passes CHECK, you will want to see whether it contains words that are not yet in the MOR lexicon. You can do this quickly by running the command

```bash
mor +xl *.cha
```

and checking the output file which will list all the words not yet recognized by MOR. It is extremely unlikely that every word in any large corpus of child language data would be listed in even the largest MOR lexicon. Therefore, users of MOR need to understand how to supplement the basic lexicons with additional entries. Before we look at the process of adding new words to the lexicon, we first need to examine the way in which entries in the disk lexicon are structured.

The disk lexicon contains truly irregular forms of a word as well as citation forms. For example, the verb “go” is stored in the disk lexicon, along with the past tense “went,” since this latter form is suppletive and does not undergo regular rules. The disk lexicon contains any number of lexical entries, stored at most one entry per line. The lexicon may be annotated with comments, which will not be processed. A comment begins with the percent sign and ends with a new line. A lexical entry consists of these parts:

1. The surface form of the word.
2. Category information about the word, expressed as a set of feature-value pairs.
Each feature-value pair is enclosed in square brackets and the full set of feature-value pairs is enclosed in curly braces. All entries must contain a feature-value pair that identifies the syntactic category to which the word belongs, consisting of the feature “scat” with an appropriate value.

3. Following the category information is information about the lemmatization of irregular forms. This information is given by having the citation form of the stem followed by the & symbol as the morpheme separator and then the grammatical morphemes it contains.

4. Finally, if the grammar is for a language other than English, you can enter the English translation of the word preceded by and followed by the = sign.

The following are examples of lexical entries:

- can  \{[scat v:aux]\}
- a  \{[scat det]\}
- an  \{[scat det]\} "a"
- go  \{[scat v] [ir +]\}
- went  \{[scat v] [tense past]\} "go&PAST"

When adding new entries to the lexicon it is usually sufficient to enter the citation form of the word, along with the syntactic category information, as in the illustration for the word “a” in the preceding examples. When working with languages other than English, you may wish to add English glosses and even special character sets to the lexicon. For example, in Cantonese, you could have this entry:

- ping4gwo2  \{[scat n]\} =apple=

To illustrate this, here is an example of the MOR output for an utterance from Cantonese:

- *CHI:  sik6 ping4gwo2 caang2 hoeng1ziu1 .
- %mor:  v|sik6=eat n|ping4gwo2=apple
  n|caang2=orange n|hoeng1ziu1=banana .

In languages that use both Roman and non-Roman scripts, such as Chinese, you may also want to add non-Roman characters after the English gloss. This can be done using this form in which the $ sign separates the English gloss from the representation in characters:

- pinyin  \{[scat x]\} "lemmatization" =gloss$characters=

MOR will take the forms indicated by the lemmatization, the gloss, and the characters and append them after the category representation in the output. The gloss should not contain spaces or the morpheme delimiters +, -, and #. Instead of spaces or the + sign, you can use the underscore character to represent compounds.

### 11.5.4 Lexicon Building

Once the file is thoroughly CHECK-ed, you are ready to make a first run of MOR to see how many words in your files are not yet in the MOR lexicon. The command is simply:

```
mor +xl *.mor
```

When MOR is run with the +xl flag, the output is a single “minilex” file with the
extension .ulx which contains templates for the lexical entries for all unknown words in a collection of files. Duplicates are removed automatically when MOR creates the .ulx file. A fragment of the output of this command might look something like this:

```
ta   {{scat ?}}
tag   {{scat ?}}
tags  {{scat ?}}
talkative  {{scat ?}}
tambourine  {{scat ?}}
```

You must then go through this file and determine whether to discard, complete, or modify these entry templates. For example, it may be impossible to decide what category “ta” belongs to without examining where it occurs in the corpus. In this example, a scan of the Sarah files in the Brown corpus (from which these examples were taken), reveals that “ta” is a variant of the infinitive marker “to”:

```
*MER:  yeah # (be)cause if it's gon (t)a be a pel it's
got ta go that way.
```

Therefore, the entry for “ta” is amended to:

```
ta   {{scat inf}} "to"
```

The corpus includes both the form “tag” and “tags.” However, because the former can be derived from the latter, it is sufficient to have just the entry for “tag” in the lexicon. The forms “talkative” and “tambourine” are low-frequency items that are not included in the standard lexicon file eng.lex. Inasmuch as these are real words, the ? should be replaced by the codes “adj” and “n”, respectively. For the example fragment given above, the resulting .ulx file should look like this:

```
ta   {{scat inf}} "to"
tag   {{scat n}}
talkative  {{scat adj}}
tambourine  {{scat n}}
```

Once all words have been coded, you need to insert each new word into one of the lexicon files. If you do not want to edit the main files, you can create new ones such as adj2.cut for all your new adjectives or vir2.cut for additional irregular verbs.

### 11.5.5 A Formal Description of the Rule Files

Users working with languages for which grammar files have already been built do not need to concern themselves with the remaining sections on MOR. However, users who need to develop grammars for new languages or who find they have to modify grammars for existing ones will need to understand how to create the two basic rule files themselves. You do not need to create a new version of the sf.cut file for special form markers. You just copy this file and give it a name such as dansf.cut, if the prefix you want to use for your language is something like “dan” for Danish.

In order to build new versions of the arules and crules files for your language, you will need to study the English files or files for a related language. For example, when you are building a grammar for Portuguese, it would be helpful to study the grammar that has already been constructed for Spanish. This section will help you understand the basic principles underlying the construction of the arules and crules.
Declarative structure

Both arules and crules are written using a simple declarative notation. The following formatting conventions are used throughout:

1. Statements are one per line. Statements can be broken across lines by placing the continuation character \ at the end of the line.
2. Comments begin with a % character and are terminated by the new line. Comments may be placed after a statement on the same line, or they may be placed on a separate line.
3. Names are composed of alphanumeric symbols, plus these characters:
   \^ & + - : \ @ . /

Both arule and crule files contain a series of rules. Rules contain one or more clauses, each of which is composed of a series of condition statements, followed by a series of action statements. In order for a clause in rule to apply, the input(s) must satisfy all condition statements. The output is derived from the input via the sequential application of all the action statements.

Both condition and action statements take the form of equations. The left hand side of the equation is a keyword, which identifies the part of the input or output being processed. The right hand side of the rule describes either the surface patterns to be matched or generated, or the category information that must be checked or manipulated.

The analyzer manipulates two different kinds of information: information about the surface shape of a word, and information about its category. All statements that match or manipulate category information must make explicit reference to a feature or features. Similarly, it is possible for a rule to contain a literal specification of the shape of a stem or affix. In addition, it is possible to use a pattern matching language in order to give a more general description of the shape of a string.

Pattern-matching symbols

The specification of orthographic patterns relies on a set of symbols derived from the regular expression (regexp) system in Unix. The rules of this system are:

1. The metacharacters are: * [ ] \ . ! All other characters are interpreted literally.
2. A pattern that contains no metacharacters will only match itself, for example the pattern “abc” will match only the string “abc”.
3. The period matches any character.
4. The asterisk * allows any number of matches (including 0) on the preceding character. For example, the pattern ‘.*’ will match a string consisting of any number of characters.
5. The brackets [ ] are used to indicate choice from among a set of characters. The pattern [ab] will match either a or b.
6. A pattern may consist of a disjunctive choice between two patterns, by use of the | symbol. For example, the pattern will match all strings which end in x, s, sh, or
7. It is possible to check that some input does not match a pattern by prefacing the entire pattern with the negation operator !.

**Variable notation**

A variable is used to name a regular expression and to record patterns that match it. A variable must first be declared in a special variable declaration statement. Variable declaration statements have the format: “\texttt{VARNAME = regular-expression}” where \texttt{VARNAME} is at most eight characters long. If the variable name is more than one character, this name should be enclosed in parenthesis when the variable is invoked. Variables are particularly important for the arules in the ar.cut file. In these rules, the negation operator is the up arrow ^, not the exclamation mark. Variables may be declared through combinations of two types of disjunction markers, as in this example for the definition of a consonant cluster in the English ar.cut file:

\[
O = [^aeiou][^aeiou][aeiou][^aeiou][aeiou][^aeiou]qu|squ
\]

Here, the square brackets contain the definition of a consonant as not a vowel and the bar or turnstile symbols separate alternative sequences of one, two, or three consonants. Then, for good measure, the patterns “qu” and “squ” are also listed as consonantal onsets. For languages that use combining diacritics and other complex symbols, it is best to use the turnstile notation, since the square bracket notation assumes single characters. In these strings, it is important not to include any spaces or tabs, since the presence of a space will signal the end of the variable.

Once declared, the variable can be invoked in a rule by using the operator $. If the variable name is longer than a single character, the variable name should be enclosed in parentheses when invoked. For example, the statement \texttt{X = .\*} declares and initializes a variable named “X.” The name X is entered in a special variable table, along with the regular expression it stands for. Note that variables may not contain other variables.

The variable table also keeps track of the most recent string that matched a named pattern. For example, if the variable X is declared as above, then the pattern $Xle will match all strings that end in “le”. In particular, the string “able” will match this pattern; “ab” will match the pattern named by “X”, and “le” will match the literal string “le”. Because the string “ab” is matched against the named pattern X, it will be stored in the variable table as the most recent instantiation of X, until another string matches X.

**Category Information Operators**

The following operators are used to manipulate category information: ADD [feature value], and DEL [feature value]. These are used in the category action statements. For example, the crule statement “RESULTCAT = ADD [num pl]” adds the feature value pair [num pl] to the result of the concatenation of two morphemes.

**Arules**

The function of the arules is to expand the entries in the disk lexicon into a larger
number of entries in the on-line lexicon. Words that undergo regular phonological or orthographic changes when combined with an affix only need to have one disk lexicon entry. The arules are used to create on-line lexicon entries for all inflectional variants. These variants are called allos. For example, the final consonant of the verb “stop” is doubled before a vowel-initial suffix, such as “-ing.” The disk lexicon contains an entry for “stop,” whereas the online lexicon contains two entries: one for the form “stop” and one for the form “stopp”.

An arule consists of a header statement, which contains the rulename, followed by one or more condition-action clauses. Each clause has a series of zero or more conditions on the input, and one or more sets of actions. Here is an example of a typical condition-action clause from the larger n-allo rule in the English ar.cut file:

```
LEX-ENTRY:
LEXSURF = $Yy
LEXCAT = [scat n]
ALLO:
ALLOSURF = $Yie
ALLOCAT = LEXCAT, ADD [allo nYb]
ALLO:
ALLOSURF = LEXSURF
ALLOCAT = LEXCAT, ADD [allo nYa]
```

This is a single condition-action clause, labeled by the header statement “LEX-ENTRY:.” Conditions begin with one of these two keywords:

1. LEXSURF matches the surface form of the word in the lexical entry to an abstract pattern. In this case, the variable declaration is $Y = .[^aeiou]$ Given this, the statement “LEXSURF = $Yy” will match all lexical entry surfaces that have a final y preceded by a nonvowel.

2. LEXCAT checks the category information given in the matched lexical item against a given series of feature value pairs, each enclosed in square brackets and separated by commas. In this case, the rule is meant to apply only to nouns, so the category information must be [scat n]. It is possible to check that a feature-value pair is not present by prefacing the feature-value pair with the negation operator !.

Variable declarations should be made at the beginning of the rule, before any of the condition-action clauses. Variables apply to all following condition-action clauses inside a rule, but should be redefined for each rule.

After the condition statements come one or more action statements with the label ALLO: In most cases, one of the action statements is used to create an allomorph and the other is used to enter the original lexical entry into the run-time lexicon. Action clauses begin with one of these three keywords:

1. ALLOSURF is used to produce an output surface. An output is a form that will be a part of the run-time lexicon used in the analysis. In the first action clause, a lexical entry surface form like “pony” is converted to “ponie” to serve as the
stem of the plural. In the second action clause, the original form “pony” is kept because the form “ALLOSURF = LEXSURF” causes the surface form of the lexical entry to be copied over to the surface form of the allo.

2. ALLOCAT determines the category of the output allos. The statement “ALLOCAT = LEXCAT” causes all category information from the lexical entry to be copied over to the allo entry. In addition, these two actions add the morphological classes such as [allo nYa] or [allo nYb] in order to keep track of the nature of these allomorphs during the application of the crules.

4. ALLOSTEM is used to produce an output stem. This action is not necessary in this example, because this rule is fully regular and produces a noninflected stem. However, the arule that converts “postman” into “postmen” uses this ALLOSTEM action:

\[
\text{ALLOSTEM} = \$Xman&PL
\]

The result of this action is the form postman&PL that is placed into the %mor line without the involvement of any of the concatenation rules.

There are two special feature types that operate to dump the contents of the arules and the lexicon into the output. These are “gen” and “proc”. The gen feature introduces its value as a component of the stem. Thus the entry [gen masc] for the Spanish word “hombre” will end up producing n|hombre&MASC. The entry [proc dim] for Chinese reduplicative verbs will end up producing v|kan4-DIM for the reduplicated form kan4kan4. These methods allow allorules to directly influence the output of MOR.

Every set of action statements leads to the generation of an additional allomorph for the online lexicon. Thus, if an arule clause contains several sets of action statements, each labeled by the header ALLO:, then that arule, when applied to one entry from the disk lexicon, will result in several entries in the online lexicon. To create the online lexicon, the arules are applied to the entries in the disk lexicon. Each entry is matched against the arules in the order in which they occur in the arules file. This ordering of arules is an extremely important feature. It means that you need to order specific cases before general cases to avoid having the general case preempt the specific case.

As soon as the input matches all conditions in the condition section of a clause, the actions are applied to that input to generate one or more allos, which are loaded into the on-line lexicon. No further rules are applied to that input, and the next entry from the disk lexicon is then read in to be processed. The complete set of arules should always end with a default rule to copy over all remaining lexical entries that have not yet been matched by some rule. This default rule must have this shape:

\[
\% \text{ default rule- copy input to output}
\]

\[
\begin{align*}
\text{RULENAME: default} \\
\text{LEX-ENTRY:} \\
\text{ALLO:}
\end{align*}
\]

**Crules**

The purpose of the crules is to allow stems to combine with affixes. In these rules, sets of conditions and actions are grouped together into if then clauses. This allows a rule
to apply to a disjunctive set of inputs. As soon as all the conditions in a clause are met, the actions are carried out. If these are carried out successfully the rule is considered to have “fired,” and no further clauses in that rule will be tried.

There are two inputs to a crule: the part of the word identified thus far, called the “start,” and the next morpheme identified, called the “next.” The best way to think of this is in terms of a bouncing ball that moves through the word, moving items from the not-yet-processed chunk on the right over to the already processed chunk on the left. The output of a crule is called the “result.” The following is the list of the keywords used in the crules:

<table>
<thead>
<tr>
<th>condition keywords</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTSURF</td>
<td>check surface of start input against some pattern</td>
</tr>
<tr>
<td>STARTCAT</td>
<td>check start category information</td>
</tr>
<tr>
<td>NEXTSURF</td>
<td>check surface of next input against some pattern</td>
</tr>
<tr>
<td>NEXTCAT</td>
<td>check next category information</td>
</tr>
<tr>
<td>MATCHCAT</td>
<td>check that start and next have the same value for all the feature-value pairs of the type specified</td>
</tr>
<tr>
<td>RESULTCAT</td>
<td>output category information</td>
</tr>
</tbody>
</table>

Here is an example of a piece of a rule that uses most of these keywords:

```
S = .*[sc]h.*[zxs] % strings that end in affricates
O = .*[^aeiou]o % things that end in o
% clause 1 - special case for "es" suffix
if
    STARTSURF = $S
    NEXTSURF = es|es
    NEXTCAT = [scat vsfx]
    MATCHCAT [allo]
then
    RESULTCAT = STARTCAT, NEXTCAT [tense], DEL [allo]
RULEPACKAGE = ()
```

This rule is used to analyze verbs that end in -es. There are four conditions that must be matched in this rule:

1. The STARTSURF is a stem that is specified in the declaration to end in an affricate. The STARTCAT is not defined.
2. The NEXTSURF is the -es suffix that is attached to that stem.
3. The NEXTCAT is the category of the suffix, which is “vsfx” or verbal suffix.
4. The MATCHCAT [allo] statement checks that both the start and next inputs have the same value for the feature allo. If there are multiple [allo] entries, all must match.

The shape of the result surface is simply the concatenation of the start and next surfaces. Hence, it is not necessary to specify this via the crules. The category information of the result is specified via the RESULTCAT statement. The statement “RESULTCAT = STARTCAT” causes all category information from the start input to be copied over to the result. The statement “NEXTCAT [tense]” copies the tense value from the NEXT to the
RESULT and the statement “DEL [allo]” deletes all the values for the category [allo].

In addition to the condition-action statements, crules include two other statements: the CTYPE statement, and the RULEPACKAGES statement. The CTYPE statement identifies the kind of concatenation expected and the way in which this concatenation is to be marked. This statement follows the RULENAME header. There are two special CTYPE makers: START and END. “CTYPE: START” is used for those rules that execute as soon as one morpheme has been found. “CTYPE: END” is used for those rules that execute when the end of the input has been reached. Otherwise, the CTYPE marker is used to indicate which concatenation symbol is used when concatenating the morphemes together into a parse for a word. According to CLAN conventions, # is used between a prefix and a stem, - is used between a stem and suffix, and ~ is used between a clitic and a stem. In most cases, rules that specify possible suffixes will start with CTYPE: -. These rules are neither start nor end rules and they insert a suffix after the stem.

Rules with CTYPE START are entered into the list of startrules. Startrules are the set of rules applied as soon as a morpheme has been recognized. In this case, the beginning of the word is considered as the start input, and the next input is the morpheme first recognized. As the start input has no surface and no category information associated with it, conditions and actions are stated only on the next input.

Rules with CTYPE END are entered into the list of endrules. These rules are invoked when the end of a word is reached, and they are used to rule out spurious parses. For the endrules, the start input is the entire word that has just been parsed, and there is no next input. Thus conditions and actions are only stated on the start input.

The RULEPACKAGES statement identifies which rules may be applied to the result of a rule, when that result is the input to another rule. The RULEPACKAGES statement follows the action statements in a clause. There is a RULEPACKAGES statement associated with each clause. The rules named in a RULEPACKAGES statement are not tried until after another morpheme has been found. For example, in parsing the input “walking”, the parser first finds the morpheme “walk,” and at that point applies the startrules. Of these startrules, the rule for verbs will be fired. This rule includes a RULEPACKAGES statement specifying that the rule which handles verb conjugation may later be fired. When the parser has further identified the morpheme “ing,” the verb conjugation rule will apply, where “walk” is the start input, and “ing” is the next input.

Note that, unlike the arules which are strictly ordered from top to bottom of the file, the crules have an order of application that is determined by their CTYPE and the way in which the RULEPACKAGES statement channels words from one rule to the next.

11.5.6 Interactive Mode

When building a grammar for a new language, it is best to begin with a paper-and-pencil analysis of the morphological system in which you lay out the various affixes of the language, the classes of stem allomorphy variations, and the forces that condition the
choices between allomorphs. This work should be guided by a good descriptive grammar of the morphology of the language. Once this work is finished, you should create a small lexicon of the most frequent words. You may want to focus on one part-of-speech at a time. For example, you could begin with the adverbs, since they are often monomorphemic. Then you could move on to the nouns. The verbs should probably come last. You can copy the sf.cut file from English and rename it.

Once you have a simple lexicon and a set of rule files, you will begin a long process of working with interactive MOR. When using MOR in the +xi or interactive mode, there are several additional options that become available in the CLAN Output window. They are:

```
word - analyze this word
:q quit- exit program
:c print out current set of crules
:d display application of a rules.
:l re-load rules and lexicon files
:h help - print this message
```

If you type in a word, such as “dog” or “perro,” MOR will try to analyze it and give you its component morphemes. If all is well, you can move on the next word. If it is not, you need to change your rules or the lexicon. You can stay within CLAN and just open these using the Editor. After you save your changes, use :l to reload and retest.

When you begin work with the grammar, you want to focus on the use of the +xi switch, rather than the analysis of large groups of files. As you begin to elaborate your grammar, you will want to start to work with sets of files. These can be real data files or else files full of test words. When you shift to working with files, you will be combining the use of interactive MOR and the +xi switch with use of the lexicon testing facility that uses +xl. As you move through this work, make copies of your MOR grammar files and lexicon frequently, because you will sometimes find that you have made a change that makes everything break and you will need to go back to an earlier stage to figure out what you need to fix. We also recommend using a fast machine with lots of memory. You will find that you are frequently reloading the grammar using the :l function. Having a fast machine will greatly speed this process.

To begin the process, start working with the sample minimal MOR grammars available from the net. These files should allow you to build up a lexicon of uninflected stems. Try to build up separate files for each of the parts of speech in your language. As you start to feel comfortable with this, you should begin to add affixes. To do this, you need to create a lexicon file, such as aff.cut. Using the technique found in unification grammars, you want to set up categories and allos for these affixes that will allow them to match up with the right stems when the crules fire. For example, you might want to call the plural a [scat nsfx] in order to emphasize the fact that it should attach to nouns. And you could give the designation [allo mdim] to the masculine diminutive suffix -ito in Spanish in order to make sure that it only attaches to masculine stems and produces a masculine output.

As you progress with your work, continually check each new rule change by entering
:l (colon followed by “l” for load) into the CLAN Output window. If you have changed something in a way that produces a syntactic violation, you will learn this immediately and be able to change it back. If you find that a method fails, you should first rethink your logic. Consider these factors:

1. Arules are strictly ordered. Maybe you have placed a general case before a specific case.
2. Crules depend on direction from the RULEPACKAGES statement.
3. There has to be a START and END rule for each part of speech. If you are getting too many entries for a word, maybe you have started it twice. Alternatively, you may have created too many allomorphs with the arules.
4. If you have a MATCHCAT allos statement, all allos must match. The operation DEL [allo] deletes all allos and you must add back any you want to keep.
5. Make sure that you understand the use of variable notation and pattern matching symbols for specifying the surface form in the arules.

However, sometimes it is not clear why a method is not working. In this case, you will want to check the application of the crules using the :c option in the CLAN Output window. You then need to trace through the firing of the rules. The most important information is often at the end of this output.

If the stem itself is not being recognized, you will need to also trace the operation of the arules. To do this, you should either use the +e option in standard MOR or else the :d option in interactive MOR. The latter is probably the most useful. To use this option, you should create a directory called testlex with a single file with the words you are working with. Then run:

```
  mor +xi +ltestlex
```

Once this runs, type :d and then :l and the output of the arules for this test lexicon will go to debug.cdc. Use your editor to open that file and try to trace what is happening there.

As you progress with the construction of rules and the enlargement of the lexicon, you can tackle whole corpora. At this point you will occasionally run the +xl analysis. Then you take the problems noted by +xl and use them as the basis for repeated testing using the +xi switch and repeated reloading of the rules as you improve them. As you build up your rule sets, you will want to annotate them fully using comments preceded by the % symbol.

11.5.7 Disambiguator Mode

Disambiguation is a special facility that is used to “clean up” the ambiguities in the %mor tier that are created by MOR. If you use the POST program, hand disambiguation is not necessary. However, when developing POST for a new language, you may find this tool useful. Toggling the Disambiguator Mode option in the Mode menu allows you to go back and forth between Disambiguator Mode and standard Editor Mode. In Disambiguator Mode, you will see each ambiguous interpretation on a %mor line broken
into its alternative possibilities at the bottom of the editor screen. The user double-clicks on the correct option and it is inserted. An ambiguous entry is defined as any entry that has the ^ symbol in it. For example, the form N|back^Prep|back is ambiguously either the noun “back” or the preposition “back.”

By default, Disambiguator Mode is set to work on the %mor tier. However, you may find it useful for other tiers as well. To change its tier setting, select the Edit menu and pull down to Options to get the Options dialog box. Set the disambiguation tier to the tier you want to disambiguate. To test all of this out, edit the sample.cha file, reset your default tier, and then type Esc-2. The editor should take you to the second %spa line which has:

```
%spa: $RES:sel:ve$DES:tes:ve
```

At the bottom of the screen, you will have a choice of two options to select. Once the correct one is highlighted, you hit a carriage return and the correct alternative will be inserted. If you find it impossible to decide between alternative tags, you can select the UND or undecided tag, which will produce a form such as “und|drin^k” for the word drink, when you are not sure whether it is a noun or a verb.

### 11.6 The Workings of MOR

When MOR runs, it breaks up morphemes into their component parts. In a relatively analytic language like English, many words require no analysis at all. However, even in English, a word like “coworkers” can be seen to contain four component morphemes, including the prefix “co”, the stem, the agential suffix, and the plural. For this form, MOR will produce the analysis: co#n|vwork-AGT-PL. This representation uses the symbols # and – to separate the four different morphemes. Here, the prefix stands at the beginning of the analysis, followed by the stem (n|work), and the two suffixes. In general, stems always have the form of a part of speech category, such as “n” for noun, followed by the vertical bar and then a statement of the stem’s lexical form.

In order to understand the functioning of the MOR grammar for English, the best place to begin is with a tour of the files inside the /english folder that you can download from the server. At the top level, you will see these files:

1. ar.cut – These are the rules that generate allomorphic variants from stems.
2. cr.cut – These are the rules that specify the possible combinations of morphemes going from left to right in an English word.
3. debug.cdc – This file holds the complete trace of an analysis of a given word by MOR. It always holds the results of the most recent analysis. It is mostly useful for people who are developing new ar.cut or cr.cut files as a way of tracing out or debugging problems with these rules.
4. docs – This is a folder containing a file of instructions on how to train POST and a list of tags and categories used in the English grammar.
5. eng.db – This is a file used by POST and should be left untouched.
6. ex.cut – This file includes analyses that are being “overgenerated” by MOR and should simply be filtered out or excluded whenever they occur.
7. lex – This is the heart of the MOR grammar. We will examine it in greater detail below.

8. posttags.cut – This is a list of the grammatical tags of English that should be included whenever running POST.

9. sf.cut – This file tells MOR how to deal with words that end with certain special form markers such as @b for babbling.

10. traintags.cut – This is a list of the tags that are used by POSTTRAIN when it creates the eng.db database.

When examining these files and others, please note that the exact shapes of the files, the word listings, and the rules will change over time. We recommend that users open up these various files to understand their contents. However, over time, the contents will diverge more and more from the names and examples given here. Still, it should be possible to trace through the same basic principles, even given these inevitable changes.

Now let us take a look at the files contained inside the /lex folder. Here, we find 72 files that break out the possible words of English into different files for each specific part of speech or compound structure. Because these distinctions are so important to the correct transcription of child language and the correct running of MOR, it is worthwhile to consider the contents of each of these various files. As the following table shows, about half of these word types involve different part of speech configurations within compounds. This analysis of compounds into their part of speech components is intended to further study of the child’s learning of compounds as well as to provide good information regarding the part of speech of the whole. The name of the compound files indicates their composition. For example the name adj+n+adj.cut indicates compounds with a noun followed by an adjective (n+adj) whose overall function is that of an adjective. In English, the part of speech of a compound is usually the same as that of the last component of the compound.
<table>
<thead>
<tr>
<th>File</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0affix.cut</td>
<td>prefixes and suffixes</td>
<td>see expanded list below</td>
</tr>
<tr>
<td>0uk.cut</td>
<td>terms local to the UK</td>
<td>fave, doofer, sixpence</td>
</tr>
<tr>
<td>adj-dup.cut</td>
<td>baby talk doubles</td>
<td>nice+nice, pink+pink</td>
</tr>
<tr>
<td>adj-ir.cut</td>
<td>irregular adjectives</td>
<td>better, furthest</td>
</tr>
<tr>
<td>adj-kidy.cut</td>
<td>adjectives with babytalk –y</td>
<td>bunchy, eaty, crawly</td>
</tr>
<tr>
<td>adj.cut</td>
<td>regular adjectives</td>
<td>tall, redundant</td>
</tr>
<tr>
<td>adj+adj+adj.cut</td>
<td>compounds</td>
<td>half+hearted, hot+crossed</td>
</tr>
<tr>
<td>adj+adj+adj(on).cut</td>
<td>compounds</td>
<td>super+duper, easy+peasy</td>
</tr>
<tr>
<td>adj+n+adj.cut</td>
<td>compounds</td>
<td>dog+eared, stir+crazy</td>
</tr>
<tr>
<td>adj+v+prep+n.cut</td>
<td>compounds</td>
<td>pay+per+view</td>
</tr>
<tr>
<td>adv-int.cut</td>
<td>intensifying adverbs</td>
<td>really, plumb, quite</td>
</tr>
<tr>
<td>adv-loc.cut</td>
<td>locative adverbs</td>
<td>north, upstairs</td>
</tr>
<tr>
<td>adv-tem.cut</td>
<td>temporal adverbs</td>
<td>tomorrow, tonight, anytime</td>
</tr>
<tr>
<td>adv.cut</td>
<td>regular adverbs</td>
<td>ajar, fast, mostly</td>
</tr>
<tr>
<td>adv+adj+adv.cut</td>
<td>compounds</td>
<td>half+off, slant+wise</td>
</tr>
<tr>
<td>adv+adj+n.cut</td>
<td>compounds</td>
<td>half+way, off+shore</td>
</tr>
<tr>
<td>adv+n+prep+n.cut</td>
<td>compounds</td>
<td>face+to+face</td>
</tr>
<tr>
<td>auxil.cut</td>
<td>auxiliaries and modals</td>
<td>should, can, are</td>
</tr>
<tr>
<td>co-cant.cut</td>
<td>Cantonese bilingual forms</td>
<td>wo, wai, la</td>
</tr>
<tr>
<td>co-voc.cut</td>
<td>vocative communicators</td>
<td>honey, dear, sir</td>
</tr>
<tr>
<td>co-rhymes.cut</td>
<td>rhymes, onomatopoeia</td>
<td>cock_a_doodle_doo</td>
</tr>
<tr>
<td>co_under.cut</td>
<td>multiword phrases</td>
<td>by_jove, gee_whiz</td>
</tr>
<tr>
<td>co.cut</td>
<td>regular communicators</td>
<td>blah, byebye, gah, no</td>
</tr>
<tr>
<td>conj.cut</td>
<td>conjunctions</td>
<td>and, although, because</td>
</tr>
<tr>
<td>det.cut</td>
<td>deictic determiners</td>
<td>this, that, the,</td>
</tr>
<tr>
<td>fil.cut</td>
<td>fillers</td>
<td>um, uh, er</td>
</tr>
<tr>
<td>n-abbrev.cut</td>
<td>abbreviations</td>
<td>c_d, t_v, w_c</td>
</tr>
<tr>
<td>n-baby.cut</td>
<td>babyl talk forms</td>
<td>passie, wawa, booboo</td>
</tr>
<tr>
<td>n-dashed.cut</td>
<td>non-compound combinations</td>
<td>cul_de_sac, seven_up</td>
</tr>
<tr>
<td>n-dup.cut</td>
<td>duplicate nouns</td>
<td>cow+cow, chick_chick</td>
</tr>
<tr>
<td>n-ir.cut</td>
<td>irregular nouns</td>
<td>children, cacti, teeth</td>
</tr>
<tr>
<td>n-loan.cut</td>
<td>loan words</td>
<td>goyim, amigo, smuck</td>
</tr>
<tr>
<td>n-plural+ant.cut</td>
<td>nouns with no singular</td>
<td>golashes, kinesics, scissors</td>
</tr>
<tr>
<td>n.cut</td>
<td>regular nouns</td>
<td>dog, corner, window</td>
</tr>
<tr>
<td>n+adj+n.cut</td>
<td>compounds</td>
<td>big+shot, cutie+pie</td>
</tr>
<tr>
<td>n+adj+v+adj.cut</td>
<td>compounds</td>
<td>merry+go+round</td>
</tr>
<tr>
<td>n+n+conj+n.cut</td>
<td>compounds</td>
<td>four+by+four, dot+to+dot</td>
</tr>
<tr>
<td>n+n+n-on.cut</td>
<td>compounds</td>
<td>quack+duck, moo+cow</td>
</tr>
<tr>
<td>n+n+n.cut</td>
<td>compounds</td>
<td>candy+bar, foot+race</td>
</tr>
<tr>
<td>n+n+novel.cut</td>
<td>compounds</td>
<td>children+bed, dog+fish</td>
</tr>
<tr>
<td>n+n+prep+det+n.cut</td>
<td>compounds</td>
<td>corn+on+the+cob</td>
</tr>
<tr>
<td>n+on+on-baby.cut</td>
<td>compounds</td>
<td>wee+wee, meow+meow</td>
</tr>
<tr>
<td>n+v+x+n.cut</td>
<td>compounds</td>
<td>jump+over+hand</td>
</tr>
<tr>
<td>n+v+n.cut</td>
<td>compounds</td>
<td>squirm+worm, snap+bead</td>
</tr>
<tr>
<td>n+prep.cut</td>
<td>compounds</td>
<td>chin+up, hide+out</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>num-ord.cut</td>
<td>ordinals</td>
<td>fourth, thirteenth</td>
</tr>
<tr>
<td>num.cut</td>
<td>cardinals</td>
<td>five, twenty</td>
</tr>
<tr>
<td>on.cut</td>
<td>onomatopoeia</td>
<td>boom, choo_choo</td>
</tr>
<tr>
<td>on+on+on.cut</td>
<td>compounds</td>
<td>cluck+cluck, knock+knock</td>
</tr>
<tr>
<td>prep.cut</td>
<td>prepositions</td>
<td>under, minus</td>
</tr>
<tr>
<td>pro.cut</td>
<td>pronouns</td>
<td>he, nobody, himself</td>
</tr>
<tr>
<td>quan.cut</td>
<td>quantifier</td>
<td>some, all, only, most</td>
</tr>
<tr>
<td>small.cut</td>
<td>assorted forms</td>
<td>not, to, xxx, yyy</td>
</tr>
<tr>
<td>v-baby.cut</td>
<td>baby verbs</td>
<td>wee, poo</td>
</tr>
<tr>
<td>v-clit.cut</td>
<td>cliticized forms</td>
<td>gonna, looka</td>
</tr>
<tr>
<td>v-dup.cut</td>
<td>verb duplications</td>
<td>eat+eat, drip+drip</td>
</tr>
<tr>
<td>v-ir.cut</td>
<td>irregular verbs</td>
<td>came, beset, slept</td>
</tr>
<tr>
<td>v.cut</td>
<td>regular verbs</td>
<td>run, take, remember</td>
</tr>
<tr>
<td>v+adj+v.cut</td>
<td>compounds</td>
<td>deep+fry, tippy+toe</td>
</tr>
<tr>
<td>v+n+v.cut</td>
<td>compounds</td>
<td>bunny+hop, sleep+walk</td>
</tr>
<tr>
<td>v+v+conj+v.cut</td>
<td>compounds</td>
<td>hide+and+seek</td>
</tr>
<tr>
<td>wh.cut</td>
<td>interrogatives</td>
<td>which, how, why</td>
</tr>
<tr>
<td>zero.cut</td>
<td>omitted words</td>
<td>0know, 0conj, 0is</td>
</tr>
</tbody>
</table>

The construction of these lexicon files involves a variety of decisions. Here are some of the most important issues to consider.

1. Words may often appear in several files. For example, virtually every noun in English can also function as a verb. However, when this function is indicated by a suffix, as in “milking” the noun can be recognized as a verb through a process of morphological derivation contained in a rule in the cr.cut file. In such cases, it is not necessary to list the word as a verb. Of course, this process fails for unmarked verbs. However, it is generally not a good idea to represent all nouns as verbs, since this tends to overgenerate ambiguity. Instead, it is possible to use the POSTMORTEM program to detect cases where nouns are functioning as bare verbs.

2. If a word can be analyzed morphologically, it should not be given a full listing. For example, since “coworker” can be analyzed by MOR into three morphemes as co#n:v|work-AGT, it should not be separately listed in the n.cut file. If it is, then POST will not be able to distinguish co#n:v|work-AGT from ncoworker.

3. In the zero.cut file, possible omitted words are listed without the preceding 0. For example, there is an entry for “conj” and “the”. However, in the transcript, these would be represented as “0conj” and “0the”.

4. It is always best to use spaces to break up word sequences that are really just combinations of words. For example, instead of transcribing 1964 as “nineteen+sixty+four”, “nineteen-sixty-four”, or “nineteen_sixty_four”, it is best to transcribe simply as “nineteen sixty four”. This principle is particularly important for Chinese, where there is a tendency to underutilize spaces, since Chinese itself is written without spaces.

5. For most languages that use Roman characters, you can rely on capitalization to force MOR to treat words as proper nouns. To understand this, take a look at the
forms in the sf.cut file at the top of the MOR directory. These various entries tell MOR how to process forms like k@l for the letter “k” or John_Paul_Jones for the famous admiral. The symbol \c indicates that a form is capitalized and the symbol \l indicates that it is lowercase.

6. Deciding how to represent compounds is a difficult matter. See the discussion in the next section.

### 11.6.1 Compounds and Complex Forms

The initial formulations of CHAT that were published in 1991 (1991), 1995 (1995), and 2000 (2000) specified no guidelines for the annotation of compounds. They only stipulated that compounds should be represented with a plus. When MOR saw a word with a plus, it simply tagged it as a noun compound. This was a big mistake, since many of the compounds tagged in this way were not common nouns. Instead, they included verbs, adjectives, proper nouns, idioms, greetings, onomatopoeia, and many other nonstandard forms. Unfortunately, once this genie had been let out of the bottle, it was very difficult to convince it to go back in. To solve this problem, we had to shift from blanket recognition of compounds to an exhaustive listing of the actual forms of possible compounds. The result of this shift is that we have now created many special compound files such as n+n+n.cut or v+n+v.cut. Fixing the forms in the database to correspond to this new, tighter standard was a huge job, perhaps even more tedious than that involved in removing main line morphemicization from the corpus. However, now that we have a full analysis of compounds, there is a much more accurate analysis of children’s learning of these forms.

In the current system, compounds are listed in the lexical files according to both their overall part of speech (X-bar) and the parts of speech of their components. However, there are seven types of complex word combinations that should not be treated as compounds.

1. **Underscored words.** The n-under.cut file includes 40 forms that resemble compounds, but are best viewed as units with non-morphemic components. For example, kool_aid and band_aid are not really combinations of morphemes, although they clearly have two components. The same is true for hi_fi and coca_cola. In general, MOR and CLAN pay little attention to the underscore character, so it can be used as needed when a plus for compounding is not appropriate. The underscore mark is particularly useful for representing the combinations of words found in proper nouns such as John_Paul_Jones, Columbia_University, or The_Beauty_and_the_Beast. As long as these words are capitalized, they do not need to be included in the MOR lexicon, since all capitalized words are taken as proper nouns in English. However, these forms cannot contain pluses, since compounds are not proper nouns. And please be careful not to overuse this form.

2. **Separate words.** Many noun-noun combinations in English should just be written out as separate words. An example would be “faucet stem assembly rubber gasket holder”. We don’t want to write this as “Faucet_stem_assembly_rubber_gasket_holder” or “faucet_stem_assembly_rubber_gasket_holder” or even
“faucet+stem+assembly+rubber+gasket+holder”. It is worth noting here that German treats all such forms as single words. This means that different conventions have to be adopted for German in order to avoid the need for exhaustive listing of the infinite number of German compound nouns.

3. **Spelling sequences.** Sequences of letter names such as “O-U-T” for the spelling of “out” are transcribed with the suffix @k, as in out@k.

4. **Acronyms.** Forms such as FBI are transcribed with underscores, as in F_B_I. Presence of the initial capital letter tells MOR to treat F_B_I as a proper noun. This same format is used for non-proper abbreviations such as c_d or d_v_d.

5. **Products.** Coming up with good forms for commercial products such as Coca-Cola is tricky. Because of the need to ban the use of the dash on the main line, we have avoided the use of the dash in these names. It is clear that they should not be compounds, as in coca+cola, and compounds cannot be capitalized, so Coca+Cola is not possible. This leaves us with the option of either coca_cola or Coca_Cola. The option coca_cola seems best, since this is not really a proper noun.

6. **Babbling and word play.** In earlier versions of CHAT and MOR, transcribers often represent sequences of babbling or word play syllables as compounds. This was done mostly because the plus provides a nice way of separating out the separate syllables in these productions. In order to make it clear that these separations are simply marked for purposes of syllabification, we now ask transcribers to use forms such as ba^ba^ga^ga@wp or choo^bung^choo^bung@o to represent these patterns.

The introduction of this more precise system for transcription of complex forms opens up additional options for programs like MLU, KWAL, FREQ, and GRASP. For MLU, compounds will be counted as single words, unless the plus sign is added to the morpheme delimiter set using the +b+ option switch. For GRASP, processing of compounds only needs to look at the part of speech of the compound as a whole, since the internal composition of the compound is not relevant to the syntax. Additionally, forms such as "faucet handle valve washer assembly" do not need to be treated as compounds, since GRASP can learn to treat sequences of nouns as complex phrases header by the final noun.

### 11.6.2 Lemmatization

Researchers are often interested in computing frequency profiles that are computed using lemmas or root forms, rather inflected forms. For example, they may want to treat "dropped" as an instance of the use of the lemma "drop." In order to perform these types of computations, the KWAL, FREQ, and GEMFREQ programs use the +s@ switch to refer to various parts of complex structures in the %mor line. If you use the +s@ switch, there must be a %mor tier present. This system recognizes the following structures on the %mor line:

<table>
<thead>
<tr>
<th>Element</th>
<th>Symbol</th>
<th>Example</th>
<th>Representation</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>#</td>
<td>unwinding</td>
<td>un#vwind-PRES</td>
<td>un#</td>
</tr>
<tr>
<td>stem</td>
<td>r</td>
<td>unwinding</td>
<td>un#vwind-PRES</td>
<td>wind</td>
</tr>
<tr>
<td>suffix</td>
<td>-</td>
<td>unwinding</td>
<td>un#vwind-PRES</td>
<td>PRES</td>
</tr>
<tr>
<td>fusion</td>
<td>&amp;</td>
<td>unwound</td>
<td>un#vwind&amp;PAST</td>
<td>PAST</td>
</tr>
</tbody>
</table>
To illustrate the use of these symbols, let us look at several possible commands. All of these commands take the form: freq +t%mor -t* filename.cha. However, in addition, they add the +s switches that are given in the second column. In these commands, the asterisk is used to distinguish across forms in the frequency count and the % sign is used to combine across forms.

<table>
<thead>
<tr>
<th>Function</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>All stems with their parts of speech, merge the rest</td>
<td>+s&quot;r-,*l,0%&quot;</td>
</tr>
<tr>
<td>Only verbs</td>
<td>+s&quot;lv&quot;</td>
</tr>
<tr>
<td>All forms of the stem &quot;go&quot;</td>
<td>+s&quot;r-go&quot;</td>
</tr>
<tr>
<td>The different parts of speech of the stem &quot;go&quot;</td>
<td>+s&quot;r-go,l,0%&quot;</td>
</tr>
<tr>
<td>The stem &quot;go&quot; only when it is a verb</td>
<td>+s&quot;r-go,lv,0%&quot;</td>
</tr>
<tr>
<td>All stems, merge the rest</td>
<td>+s&quot;r,0%&quot;</td>
</tr>
</tbody>
</table>

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

11.6.3 Errors and Replacements

Transcriptions on the main line have to serve two, sometimes conflicting (Edwards, 1992), functions. On the one hand, they need to represent the form of the speech as actually produced. On the other hand, they need to provide input that can be used for morphosyntactic analysis. When words are pronounced in their standard form, these two functions are in alignment. However, when words are pronounced with phonological or morphological errors, it is important to separate out the actual production from the morphological target. This can be done through a system of main line tagging of errors. This system largely replaces the coding of errors on a separate %err line, although that form is still available, if needed. The form of the newer system is illustrated here:

*CHI: him [* case] ated [: ate] [* +ed-sup] a f(l)ower and a pun [: bun].

For the first error, there is no need to provide a replacement, since MOR can process "him" as a pronoun. However, since the second error is not a real word form, the replacement is necessary in order to tell MOR how to process the form. The third error is just an omission of "l" from the cluster and the final error is a mispronunciation of the initial consonant. Phonological errors are not coded here, since that level of analysis is best conducted inside the Phon program (Rose et al., 2005).

11.6.4 Affixes and Control Features

To complete our tour of the MOR lexicon for English, we will take a brief look at the 0affix.cut file, as well some additional control features in the other lexical files. The responsibility of processing inflectional and derivational morphology is divided across these three files. Let us first look at a few entries in the 0affix.cut file.
1. This file begins with a list of prefixes such as “mis” and “semi” that attach either to nouns or verbs. Each prefix also has a permission feature, such as [allow mis]. This feature only comes into play when a noun or verb in n.cut or v.cut also has the feature [pre no]. For example, the verb “test” has the feature [pre no] included in order to block prefixing with “de-” to produce “detest” which is not a derivational form of "test". At the same time, we want to permit prefixing with “re-”, the entry for “test” has [pre no][allow re]. Then, when the relevant rule in cr.cut sees a verb following “re-” it checks for a match in the [allow] feature and allows the attachment in this case.

2. Next we see some derivational suffixes such as diminutive –ie or agential –er. Unlike the prefixes, these suffixes often change the spelling of the stem by dropping silent e or doubling final consonants. The ar.cut file controls this process, and the [allo x] features listed there control the selection of the correct form of the suffix.

3. Each suffix is represented by a grammatical category in parentheses. These categories are taken from a typologically valid list given in the CHAT Manual.

4. Each suffix specifies the grammatical category of the form that will result after its attachment. For suffixes that change the part of speech, this is given in the scat, as in [scat adj:n]. Prefixes do not change parts of speech, so they are simply listed as [scat pfx] and use the [pcat x] feature to specify the shape of the forms to which they can attach.

5. The long list of suffixes concludes with a list of cliticized auxiliaries and reduced main verbs. These forms are represented in English as contractions. Many of these forms are multiply ambiguous and it will be the job of POST to choose the correct reading from among the various alternatives.

Outside of the 0affix.cut file, in the various other *.cut lexical files, there are several control features that specify how stems should be treated. One important set includes the [comp x+x] features for compounds. These features control how compounds will be unpacked for formatting on the %mor line. Irregular adjectives in adj-ir.cut have features specifying their degree as comparative or superlative. Irregular nouns have features controlling the use of the plural. Irregular verbs have features controlling consonant doubling [gg +] and the formation of the perfect tense.

11.6.5 Building MOR Grammars

So far, this discussion of the MOR grammar for English has avoided an examination of the ar.cut and cr.cut files. It is true that users of English MOR will seldom need to tinker with these files. However, serious students of morphosyntax need to understand how MOR and POST operate. In order to do this, they have to understand how the ar.cut and cr.cut files work. Fortunately, for English at least, these rule files are not too complex. The relative simplicity of English morphology is reflected in the fact that the ar.cut file for English has only 391 lines, whereas the same file for Spanish has 3172 lines. In English, the main patterns involve consonant doubling, silent –e, changes of y to i, and irregulars like “knives” or “leaves.” The rules use the spelling of final consonants and vowels to predict these various allomorphic variations. Variables such as $V or $C are set up at the beginning of the file to refer to vowels and consonants and then the rules
use these variables to describe alternative lexical patterns and the shapes of allomorphs. For example the rule for consonant doubling takes this shape:

LEX-ENTRY:
LEXSURF = $O$V$C$
LEXCAT = [scat v], ![tense OR past perf], ![gem no] % to block putting
ALLO:
ALLOSURF = $O$V$C$C
ALLOCAT = LEXCAT, ADD [allo vHb]
ALLO:
ALLOSURF = LEXSURF
ALLOCAT = LEXCAT, ADD [allo vHa]

Here, the string $O$V$C$ characterizes verbs like “bat” that end with vowels followed by consonants. The first allo will produce words like “battling” or “batter” and the second will give a stem for “bats” or “bat”. A complete list of allomorphy types for English is given in the file engcats.cdc in the /docs folder in the MOR grammar.

When a user types the “mor” command to CLAN, the program loads up all the *.cut files in the lexicon and then passes each lexical form past the rules of the ar.cut file. The rules in the ar.cut file are strictly ordered. If a form matches a rule, that rule fires and the allomorphs it produces are encoded into a lexical tree based on a “trie” structure. Then MOR moves on to the next lexical form, without considering any additional rules. This means that it is important to place more specific cases before more general cases in a standard bleeding relation. There is no “feeding” relation in the ar.cut file, since each form is shipped over to the tree structure after matching.

The other “core” file in a MOR grammar is the cr.cut file that contains the rules that specify pathways through possible words. The basic idea of rules or concatenation or continuation rules is taken from Hausser’s (1999) left-associative grammar which specifies the shape of possible “continuations” as a parser moves from left to right through a word. Unlike the rules of the ar.cut file, the rules in the cr.cut file are not ordered. Instead, they work through a “feeding” relation. MOR goes through a candidate word from left to right to match up the current sequence with forms in the lexical trie TREE?? structure. When a match is made, the categories of the current form become a part of the STARTCAT. If the STARTCAT matches up with the STARTCAT of one of the rules in cr.cut, as well as satisfying some additional matching conditions specified in the rule, then that rule fires. The result of this firing is to change the shape of the STARTCAT and to then thread processing into some additional rules. For example, let us consider the processing of the verb “reconsidering.” Here, the first rule to fire is the specific-vpfx-start rule which matches the fact that “re-” has the feature [scat pfx] and [pcat v]. This initial recognition of the prefix then threads into the specific-vpfx-verb rule that requires the next item have the feature [scat v]. This rule has the feature CTYPE # which serves to introduce the # sign into the final tagging to produce re#part|consider-PRESP. After the verb “consider” is accepted, the RULEPACKAGE tells MOR to move on to three other rules: v-conj, n:v-deriv, and adj:v-deriv. Each of these rules can be viewed as a separate thread out of the specific-vpfx-verb rule. At this point in processing the word, the remaining orthographic material is “-ing”. Looking at the 0affix.cut file,
we see that “ing” has three entries: [scat part], [scat v:n], and [scat n:gerund]. One of the pathways at this point leads through the v-conj rule. Within v-conj, only the fourth clause fires, since that clause matches [scat part]. This clause can lead on to three further threads, but, since there is no further orthographic material, there is no NEXTCAT for these rules. Therefore, this thread then goes on to the end rules and outputs the first successful parse of “reconsidering.” The second thread from the specific-vpfx-verb rule leads to the n:v-deriv rule. This rule accepts the reading of “ing” as [scat n:gerund] to produce the second reading of “reconsidering”. Finally, MOR traces the third thread from the specific-vpfx-verb rule which leads to adj:v-deriv. This route produces no matches, so processing terminates with this result:

Result: re#partconsider-PRESP^re#n:gerundconsider-GERUND

Later, POST will work to choose between these two possible readings of “reconsidering” on the basis of the syntactic context. As we noted earlier, when “reconsidering” follows an auxiliary (“is eating”) or when it functions adjectivally (“an eating binge”), it is treated as a participle. However, when it appears as the head of an NP (“eating is good for you”), it is treated as a gerund. Categories and processes of this type can be modified to match up with the requirements of the GRASP program to be discussed below.

The process of building ar.cut and cr.cut files for a new language involves a slow iteration of lexicon building with rule building. During this process, and throughout work with development of MOR, it is often helpful to use MOR in its interactive mode by typing: mor +xi . When using MOR in this mode, there are several additional options that become available in the CLAN Output window. They are:

word - analyze this word
:q quit- exit program
:c print out current set of rules
:d display application of a rules.
:l re-load rules and lexicon files
:h help - print this message

If you type in a word, such as “dog” or “perro,” MOR will try to analyze it and give you its component morphemes. If all is well, you can move on the next word. If it is not, you need to change your rules or the lexicon. You can stay within CLAN and just open these using the Editor. After you save your changes, use :l to reload and retest the word again.

The problem with building up a MOR grammar one word at a time like this is that changes that favour the analysis of one word can break the analysis of other words. To make sure that this is not happening, it is important to have a collection of test words that you continually monitor using mor +xl. One approach to this is just to have a growing set of transcripts or utterances that can be analyzed. Another approach is to have a systematic target set configured not as sentences but as transcripts with one word in each sentence. An example of this approach can be found in the /verbi folder in the Italian MOR grammar. This folder has one file for each of the 106 verbal paradigms of the Berlitz Italian Verb Handbook (2005). That handbook gives the full paradigm of one “leading” verb for each conjugational type. We then typed all of the relevant forms into CHAT files. Then, as we built up the ar.cut file for Italian, we designed allo types using
features that matched the numbers in the Handbook. In the end, things become a bit more complex in Spanish, Italian, and French.

1. The initial rules of the ar.cut file for these languages specify the most limited and lexically-bound patterns by listing almost the full stem, as in $Xdice for verbs like “dicere”, “predicere” or “benedicere” which all behave similarly, or “nuoce” which is the only verb of its type.

2. Further in the rule list, verbs are listed through a general phonology, but often limited to the presence of a lexical tag such as [type 16] that indicates verb membership in a conjugational class.

3. Within the rule for each verb type, the grammar specifies up to 12 stem allomorph types. Some of these have the same surface phonology. However, to match up properly across the paradigm, it is important to generate this full set. Once this basic grid is determined, it is easy to add new rules for each additional conjugational type by a process of cut-and-paste followed by local modifications.

4. Where possible, the rules are left in an order that corresponds to the order of the conjugational numbers of the Berlitz Handbook. However, when this order interferes with rule bleeding, it is changed.

5. Perhaps the biggest conceptual challenge is the formulation of a good set of [allo x] tags for the paradigm. The current Italian grammar mixes together tags like [allo vv] that are defined on phonological grounds and tags like [allo vpart] that are defined on paradigmatic grounds. A more systematic analysis would probably use a somewhat larger set of tags to cover all tense-aspect-mood slots and use the phonological tags as a secondary overlay on the basic semantic tags.

6. Although verbs are the major challenge in Romance languages, it is also important to manage verbal clitics and noun and adjectives plurals. In the end, all nouns must be listed with gender information. Nouns that have both masculine and feminine forms are listed with the feature [anim yes] that allows the ar.cut file to generate both sets of allomorphs.

7. Spanish has additional complexities involving the placement of stress marks for infinitives and imperatives with suffixed clitics, such as dámelo. Italian has additional complications for forms such as “nello” and the various pronominal and clitic forms.

To begin the process, start working with the sample “minMOR” grammars available from the net. These files should allow you to build up a lexicon of uninflected stems. Try to build up separate files for each of the parts of speech in your language. As you start to feel comfortable with this, you should begin to add affixes. To do this, you need to create a lexicon file for affixes, such as affix.cut. Using the technique found in unification grammars, you want to set up categories and allos for these affixes that will allow them to match up with the right stems when the crules fire. For example, you might want to assign [scat nsfx] to the noun plural suffix in order to emphasize the fact that it should attach to nouns. And you could give the designation [allo mdim] to the masculine diminutive suffix -ito in Spanish in order to make sure that it only attaches to masculine stems and produces a masculine output.

As you progress with your work, continually check each new rule change by entering
:l (colon followed by “l” for load) into the CLAN Output window and then testing some crucial words. If you have changed something in a way that produces a syntactic violation, you will learn this immediately and be able to change it back. If you find that a method fails, you should first rethink your logic. Consider these factors:

1. Arules are strictly ordered. Maybe you have placed a general case before a specific case.
2. Crules depend on direction from the RULEPACKAGES statement. Perhaps you are not reaching the rule that needs to fire.
3. There has to be a START and END rule for each part of speech. If you are getting too many entries for a word, maybe you have started it twice. Alternatively, you may have created too many allomorphs with the arules.
4. Possibly, you form is not satisfying the requirements of the end rules. If it doesn’t these rules will not “let it out.”
5. If you have a MATCHCAT allos statement, all allos must match. The operation DEL [allo] deletes all allos and you must add back any you want to keep.
6. Make sure that you understand the use of variable notation and pattern matching symbols for specifying the surface form in the arules.

However, sometimes it is not clear why a method is not working. In this case, you will want to check the application of the crules using the :c option in the CLAN Output window. You then need to trace through the firing of the rules. The most important information is often at the end of this output.

If the stem itself is not being recognized, you will need to also trace the operation of the arules. To do this, you should either use the +e option in standard MOR or else the :d option in interactive MOR. The latter is probably the most useful. To use this option, you should create a directory called testlex with a single file with the words you are working with. Then run: mor +xi +ltestlex

Once this runs, type :d and then :l and the output of the arules for this test lexicon will go to debug.cdc. Use your editor to open that file and try to trace what is happening there.

As you progress with the construction of rules and the enlargement of the lexicon, you can tackle whole corpora. At this point you will occasionally run the +xl analysis. Then you take the problems noted by +xl and use them as the basis for repeated testing using the +xi switch and repeated reloading of the rules as you improve them. As you build up your rule sets, you will want to annotate them fully using comments preceded by the % symbol.

11.7 Using MOR with a New Corpus

Because the English MOR grammar is stable and robust, the work of analyzing a new corpus seldom involves changes to the rules in the ar.cut or cr.cut files. However, a new English corpus is still likely to need extensive lexical clean up before it is fully recognized by MOR. The unrecognized words can be identified quickly by running this command:
mor +xl *.cha
This command will go through a collection of files and output a single file “mini lexicon” of unrecognized words. The output is given the name of the first file in the collection. After this command finishes, open up the file and you will see all the words not recognized by MOR. There are several typical reasons for a word not being recognized:

1. It is misspelled.
2. The word should be preceded by an ampersand (&) to block look up through MOR. Specifically, incomplete words should be transcribed as &text so that the ampersand character can block MOR look up. Similarly, sounds like laughing can be transcribed as &=laughs to achieve the same effect.
3. The word should have been transcribed with a special form marker, as in bobo@o or bo^bo@o for onomatopoeia. It is impossible to list all possible onomatopoeic forms in the MOR lexicon, so the @o marker solves this problem by telling MOR how to treat the form. This approach will be needed for other special forms, such as babbling, word play, and so on.
4. The word was transcribed in “eye-dialect” to represent phonological reductions. When this is done, there are two basic ways to allow MOR to achieve correct lookup. If the word can be transcribed with parentheses for the missing material, as in “(be)cause”, then MOR will be happy. This method is particularly useful in Spanish and German. Alternatively, if there is a sound substitution, then you can transcribe using the [: text] replacement method, as in “pittie [: kittie]”.
5. You should treat the word as a proper noun by capitalizing the first letter. This method works for many languages, but not in German where all nouns are capitalized and not in Asian languages, since those languages do not have systems for capitalization.
6. The word should be treated as a compound, as discussed in the previous section.
7. The stem is in MOR, but the inflected form is not recognized. In this case, it is possible that one of the analytic rules of MOR is not working. These problems can be reported to me at macw@cmu.edu.
8. The stem or word is missing from MOR. In that case, you can create a file called something like 0add.cut in the /lex folder of the MOR grammar. Once you have accumulated a collection of such words, you can email them to me for permanent addition to the lexicon.

Some of these forms can be corrected during the initial process of transcription by running CHECK. However, others will not be evident until you run the mor +xl command and get a list of unrecognized words. In order to correct these forms, there are basically two possible tools. The first is the KWAL program built in to CLAN. Let us say that your filename.ulx.cex list of unrecognized words has the form “cuaght” as a misspelling of “caught.” Let us further imagine that you have a single collection of 80 files in one folder. To correct this error, just type this command into the Commands window:

    kwal *.cha +scuaght

KWAL will then send input to your screen as it goes through the 80 files. There may
be no more than one case of this misspelling in the whole collection. You will see this as the output scrolls by. If necessary, just scroll back in the CLAN Output window to find the error and then triple click to go to the spot of the error and then retype the word correctly.

For errors that are not too frequent, this method works fairly well. However, if you have made some error consistently and frequently, you may need stronger methods. Perhaps you transcribed “byebye” as “bye+bye” as many as 60 times. In this case, you could use the CHSTRING program to fix this, but a better method would involve the use of a Programmer’s Editor system such as BBEdit for the Mac or Epsilon for Windows. Any system you use must include an ability to process Regular Expressions (RegExp) and to operate smoothly across whole directories at a time. However, let me give a word of warning about the use of more powerful editors. When using these systems, particularly at first, you may make some mistakes. Always make sure that you keep a backup copy of your entire folder before each major replacement command that you issue.

Once you have succeeded in reducing the context of the minilex to zero, you are ready to run a final pass of MOR. After that, if there is a .db file in the MOR grammar for your language, you can run POST to disambiguate your file. After disambiguation, you should run CHECK again. There may be some errors if POST was not able to disambiguate everything. In that case, you would either need to fix MOR or else just use CLAN’s disambiguate tier function (Esc-2) to finish the final stages of disambiguation.

11.8 MOR for Bilingual Corpora

It is easy to use MOR and POST to process bilingual corpora. A good sample application of this method is for the transcripts collected by Virginia Yip and Stephen Matthews from Cantonese-English bilingual children in Hong Kong. In these corpora, parents, caretakers, and children often switch back and forth between the two languages. In order to tell MOR which grammar to use for which utterances, each sentence must be clearly identified for language. It turns out that this is not too difficult to do. First, by the nature of the goals of the study and the people conversing with the child, certain files are typically biased toward one language or the other. In the YipMatthews corpus, English is the default language in folders such as SophieEng or TimEng and Cantonese is the default in folders such as SophieCan and TimCan. To mark this in the files in which Cantonese is predominant, the @Languages tier has this form:

@Language: yue, eng

In the files in which English is predominant, on the other hand, the tier has this form:

@Language: eng, yue

The programs then assume that, by default, each word in the transcript is in the first listed language. This default can be reversed in two ways. First, within the English files, the precode [- yue] can be placed at the beginning of utterances that are primarily in
Cantonese. If single Cantonese words are used inside English utterances, they are marked with the special form marker @s. If an English word appears within a Cantonese sentence marked with the [- yue] precode, then the @s code means that the default for that sentence (Chinese) is now reversed to the other language (English). For the files that are primarily in Cantonese, the opposite pattern is used. In those files, English sentences are marked as [- eng] and English words inside Cantonese are marked by @s. This form of marking preserves readability, while still making it clear to the programs which words are in which language. If it is important to have each word explicitly tagged for language, the –l switch can be used with CLAN programs such as KWAL, COMBO, or FIXIT to insert this more verbose method of language marking.

To minimize cross-language listing, it was also helpful to create easy ways of representing words that were shared between languages. This was particularly important for the names of family members or relation names. For example, the Cantonese form 姐姐 for “big sister” can be written in English as Zeze, so that this form can be processed correctly as a proper noun address term. Similarly, Cantonese has borrowed a set of English salutations such as “byebye” and “sorry” which are simply added directly to the Cantonese grammar in the co-eng.cut file.

Once these various adaptations and markings are completed, it is then possible to run MOR in two passes on the corpus. By default, MOR excludes lines marked with the form [- *] at the beginning. So, this means that, for the English corpora, the steps are:

1. Set the MOR library to English and run: mor *.cha +1
2. Disambiguate the results with: post *.cha +1
3. Run CHECK to check for problems.
4. Set the MOR library to Cantonese and run: mor +s”[- yue]” *.cha +1
5. Disambiguate the results with: post *.cha +1
6. Run CHECK to check for problems.

To illustrate the result of this process, here is a representative snippet from the te951130.cha file in the /TimEng folder. Note that the default language here is English and that sentences in Cantonese are explicitly marked as [- yue].

*LIN: where is grandma first, tell me ?
%mor: adv:whlwhr vbe nlgrandma advlfirst vtell prolmeme ?
*LIN: well, what's this ?
%mor: colwell pro:whlwhat~v:be pro:demlthi:is ?
*CHI: [- yue] xxx 呢個唔夠架。
%mor: unklxxx detnli1=thi:is cllgo3=cl neglm4=not advlgau3=enough
sfp|gaa3=sfp.
*LIN: [- yue] 呢個唔夠.
%mor: detnli1=thi:is cllgo3=cl neglm4=not advlgau3=enough.
*LIN: <what does it mean> >> ?
%mor: pro:whlwhat v:auxldo prolt vlmean ?

Currently, this type of analysis is possible whenever MOR grammars exist for both languages, as would be the case for Japanese-English, Spanish-French, Putonghua-
Cantonese, or Italian-Chinese bilinguals.

### 11.9 POST

POST was written by Christophe Parisse of INSERM, Paris for the purpose of automatically disambiguating the output of MOR. The POST package is composed of four CLAN commands: POST, POSTTRAIN, POSTLIST, and POSTMOD. POST is the command that runs the disambiguator. POST uses a database that contains information about syntactic word order. Databases are created and maintained by POSTTRAIN and can be dumped in a text file by POSTLIST. POSTMODRULES is a utility for modifying Brill rules. In this section, we describe the use of the POST command.

In order to use POST, you must have a database of disambiguation rules appropriate for your language. This file, called post.db is found in the relevant MOR grammar folder that you can download from http://childes.psy.cmu.edu/morgrams. There are also POST databases now for Chinese, Japanese, Spanish, and English. As our work with POST progresses, we will make these available for additional languages. To run POST, you can use this command format:

```
post *.cha
```

This command assumes the default values of the +f, +d, and +s switches described below.

The accuracy of disambiguation by POST for English will be above 95 percent. This means that there will be some errors. To make the most conservative use of POST, you may wish to use the +s2 switch.

The options for POST are:

- **-b** do not use Brill rules (they are used by default)
- **+bs** use a slower but more thorough version of Brill's rules analysis.
- **+c** output all affixes (default)
- **+cF** output the affixes listed in file F and post.db. If there is a posttags.cut file, then it is used by default as if the +cposttags.cut switch were being used.
- **-c** output only the affixes defined during training with POSTTRAIN
- **-cF** omit the affixes in file F, but not the affixers defined during training with POSTTRAIN
- **+dF** use POST database file F (default is "post.db"). This file must have been created by POSTTRAIN. If you do not use this switch, POST will try to locate a file called post.db in either the current working directory or your MOR library directory.
- **+e[1,2]c** this option is a complement to the option +s2 and +s3 only. It allows you to change the separator used (+e1c) between the different solutions, (+e2c) before
the information about the parsing process. (c can be any character). By default, the separator for +e1 is # and for +e2, the separator is /.

+f send output to file derived from input file name. If you do not use this switch, POST will create a series of output files named *.pst.

+FF send output to file F. This switch will change the extension to the output files.

-f send output to the screen

+lm reduce memory use (but longer processing time)

+IN when followed by a number the +I switch controls the number of output lines

+unk tries to process unknown words.

+N N=0 (default) replace ambiguous %mor lines with disambiguated ones
N=1 keep ambiguous %mor lines and add disambiguated %pos lines.
N=2 output as in N=1, but with slashes marking undecidable cases.
N=3 keep ambiguous %mor lines and add %pos lines with debugging info.
N=4 inserts a %nob line before the %mor/%pos line that presents the results of the analysis without using Brill rules.
N=5 outputs results for debugging POST grammars.
N=6 complete outputs results for debugging POST grammars.
With the options +s0 and +s1, only the best candidate is outputted. With option +s2, second and following candidates may be outputted, when the disambiguation process is not able to choose between different solutions with the most probable solution displayed first. With option +s3, information about the parsing process is given in three situations: processing of unknown words (useful for checking these words quickly after the parsing process), processing of unknown rules and no correct syntactic path obtained (usually corresponds to new grammatical situations or typographic errors).

+tS include tier code S
-tS exclude tier code S

11.10 POSTLIST

POSTLIST provides a list of tags used by POST. It is run on the *.db database file. The options for POSTLIST are as follows:

+dF this gives the name of the database to be listed (default value: ‘eng.db’).
+fF specify name of result file to be F.
+m outputs all the matrix entries present in the database.
+r outputs all the rules present in the database.
+rb outputs rule dictionary for the Brill tagger.
+rn outputs rule dictionary for the Brill tagger in numerical order.
+t outputs the list of all tags present in the database.
+w outputs all the word frequencies gathered in the database.
+wb outputs word dictionary for the Brill tagger.

If none of the options is selected, then general information about the size of the database is outputted.

Multicat categories provide a way to pack categories together so as to create artificial categories that have a longer context (four words instead of three) - this makes some sense from the linguistic point of view, it is a way to consider that clitics are in fact near-flexions and that the grammatical values is included in the word (and that in fact 'he is' is to be considered as different from 'he's', which is oral language may not be false). However if I had to redo all this, I would say the clitic / flexion distinction (whatever the theoretical interpretation) should in fact be handled at MOR level, not at POST level. POST should be get the same normalized forms, not hardcode whether they are different or dissimilar. This would be more language independent.

+r The +r option outputs rules using the following conventions.
1. pct (punctuation) indicates beginning or end of the sentence.
2. multicat/ indicates that the following categories are options
3. the numbers indicate the numbers of contexts for a particular member of the multicat set, followed by the numbers of occurrences in that context
4. n:gerund|play-GERUND^part|play-PREP^v:n|play-PREP => 3 [0,6,0] means that play has 3 potential categories n:gerund|play-GERUND and part|play-PREP and v:n|play-PREP and that it was found 6 time in the second category in the training corpus.

+x should be removed

11.11 POSTMODRULES
This program outputs the rules used by POST for debugging rules. POSTMODRULES is used to check and modify Brill rules after initial training.

11.12 POSTMORTEM
This program relies on a dictionary file called postmortem.cut to alter the part-of-speech tags in the %mor line after the final operation of MOR and POST. The use of this program is restricted to cases of extreme part-of-speech extension, such as using color names as nouns or common nouns as verbs. Here is an example of some lines in a postmortem.cut file

```
det adj v => det n v
det adj $e => det n $e
```
Here, the first line will change a sequence such as “the red is” from “det adj v” to “det n v”. The second line will change “det adj” to “det n” just in the case that the adjective is at the end of the sentence. The symbol $e$ represents the end of the utterance.

POSTMORTEM uses the +a switch to run in three possible modes. If no +a switch is used, then it does all replacements automatically. Second, if you use the +a switch, then it inserts the new string after the old one and you can then disambiguate the whole file by hand using escape-2. Third, if you use the +a1 switch, it will work interactively and you can choose whether or not to make each replacement on a case by case basis.

11.13 POSTTRAIN

POSTTRAIN was written by Christophe Parisse of INSERM, Paris. In order to run POST, you need to create a database file for your language. For several languages, this has already been done. If there is no POST database file for your language or your subject group, you can use the POSTTRAIN program to create this file. The default name for this file is eng.db. If you are not working with English, you should choose some other name for this file. Before running POSTTRAIN, you should take these steps:

1. You should specify a set of files that will be your POSTTRAIN training files. You may wish to start with a small set of files and then build up as you go.
2. You should verify that all of your training files pass CHECK.
3. Next, you should run MOR with the +xl option to make sure that all words are recognized.
4. You then run MOR on your training files. This will produce an ambiguous %mor line.
5. Now you open each file in the editor and use the Esc-2 command to disambiguate the ambiguous %mor line.
6. Once this is done for a given file, using the Query-Replace function to rename %mor to %trn.
7. After you have created a few training files or even after you have only one file, run MOR again.
8. Now you can run POSTTRAIN with a command like this:

   posttrain +c +o0err.cut *.cha

9. Now, take a look at the 0err.cut file to see if there are problems. If not, you can test out your POST file using POST. If the results seem pretty good, you can shift to eye-based evaluation of the disambiguated line, rather than using Esc-2. Otherwise, stick with Esc-2 and create more training data. Whenever you are happy with a disambiguated %mor line in a new training file, then you can go ahead and rename it to %trn.
10. The basic idea here is to continue to improve the accuracy of the %trn line as a way of improving the accuracy of the .db POST database file.

When developing a new POST database, you will find that eventually you need to repeatedly cycle through a standard sets of commands while making continual changes to the input data. Here is a sample sequence that uses the defaults in POST and POSTTRAIN:
In these commands, the +1 must be used carefully, since it replaces the original. If a program crashes or exits while running with +1, the original can be destroyed, so make a backup of the whole directory first before running +1. TRNFIX can be used to spot mismatches between the %trn and %mor lines.

The options for POSTTRAIN are:

+af  train word frequencies even on utterances longer than length 3.
+bf  extended learning using Brill's rules
-bf  Brill's rules training only
+b0F append output of Brill rule training to file F (default: send it to screen)
+bN  parameter for Brill rules
     1- means normal Brill rules are produced (default)
     2- means only lexical rules are produced
     3- same as +b1, but eliminates rules redundant with binary rules
     4- same as +b2, but eliminates rules redundant with binary rules
+b1N threshold for Brill rules (default=2). For example, if the value is 2, a rule should correct 3 errors to be considered useful. To generate all possible rules, use a threshold of 0.
+cf  create new POST database file with the name post.db (default)
+cF  create new POST database file with the name F
-cf  add to an existing version of post.db
-cF  add to an existing POST database file with the name F
+eF  the affixes and stems in file F are used for training. The default name of this file is tags.cut. So, if you want to add stems for the training, but still keep all affixes, you will need to add all the affixes explicitly to this list. You must use the +c switch when using +e.
-e   No specific file of affixes and stems is used for training. (This is the default, unless a tags.cut file is present.)
+mN  load the disambiguation matrices into memory (about 700K)
     N=0 no matrix training
     N=2 training with matrix of size 2
     N=3 training with matrix of size 3
     N=4 training with matrix of size 4 (default)
+oF  append errors output to file F (default: send it to screen)
+sN  This switch has three forms
     N=0 default log listing mismatches between the %trn and %mor line.
     N=1 similar output in a format designed more for developers.
     N=2 complete output of all date, including both matches and mismatches
+tS  include tier code S
-tS  exclude tier code S
     +/-tS#Target_Child - select target child's tiers
+/-@id="*|Mother|*" - select mother's tiers

+x use syntactic category suffixes to deal with stem compounds

When using the default switch form of the error log, lines that begin with @ indicate that the %trn and %mor had different numbers of elements. Lines that do not begin with @ represent simple disagreement between the %trn and the %mor line in some category assignment. For example, if %mor has pro:dem^pro:exist and %trn has co three times. Then +s0 would yield: 3 there co (3 {1} pro:dem (2} pro:exist).

By default, POSTTRAIN uses all the affixes in the language and none of the stems. If you wish to change this behavior, you need to create a file with your grammatical names for prefixes and suffixes or stem tags. This file can be used by both POSTTRAIN and POST. However, you may wish to create one file for use by POSTTRAIN and another for use by POST.

The English POST disambiguator currently achieves over 95% correct disambiguation. We have not yet computed the levels of accuracy for the other disambiguators. However, the levels may be a bit better for inflectional languages like Spanish or Italian. In order to train the POST disambiguator, we first had to create a hand-annotated training set for each language. We created this corpus through a process of bootstrapping. Here is the sequence of basic steps in training.

1. First run MOR on a small corpus and used the Esc-2 hand disambiguation process to disambiguate.
2. Then rename the %mor line in the corpus to %trn.
3. Run MOR again to create a separate %mor line.
4. Run POSTTRAIN with this command: posttrain +c +o0err.cut +x *.cha
5. This will create a new post.db database.
6. You then need to go through the 0errors.cut file line by line to eliminate each mismatch between your %trn line and the codes of the %mor line. Mismatches arise primarily from changes made to the MOR codes in between runs of MOR.
7. Before running POST, make sure that post.db is in the right place. The default location is in the MOR library, next to ar.cut and cr.cut. However, if post.db is not there, POST will look in the working directory. So, it is best to make sure it is located in the MOR library to avoid confusion.
8. Disambiguate the MOR line with: post *.cha +1
9. Compare the results of POST with your hand disambiguation using: trnfix *.cha

When using TRNFIX, sometimes the %trn will be at fault and sometimes %mor will be at fault. You can only fix the %trn line. To fix the %mor results, you just have to keep on compiling more training data by iterating the above process. As a rule of thumb, you eventually want to have at least 5000 utterances in your training corpus. However, a corpus with 1000 utterances will be useful initially.

During work in constructing the training corpus for POSTTRAIN, you will eventually bump into some areas of English grammar where the distinction between parts of speech is difficult to make without careful specification of detailed criteria. We can identify three
areas that are particularly problematic in terms of their subsequent effects on GR (grammatical relation) identification:

1. **Adverb vs. preposition vs. particle.** The words about, across, “after”, away, back, down, in, off, on, out, over, and up belong to three categories: ADVerb, PREPosition and ParTicLe. In practice, it is usually impossible to distinguish a particle from an adverb. Therefore, we only distinguish adverbs from prepositions. To distinguish these two, we apply the following criteria. First, a preposition must have a prepositional object. Second, a preposition forms a constituent with its noun phrase object, and hence is more closely bound to its object than an adverb or a particle. Third, prepositional phrases can be fronted, whereas the noun phrases that happen to follow adverbs or particles cannot. Fourth, a manner adverb can be placed between the verb and a preposition, but not between a verb and a particle.

2. **Verb vs. auxiliary.** Distinguishing between Verb and AUXiliary is especially tricky for the verbs be, do and have. The following tests can be applied. First, if the target word is accompanied by a nonfinite verb in the same clause, it is an auxiliary, as in I have had enough or I do not like eggs. Another test that works for these examples is fronting. In interrogative sentences, the auxiliary is moved to the beginning of the clause, as in have I had enough? and do I like eggs? whereas main verbs do not move. In verb-participle constructions headed by the verb be, if the participle is in the progressive tense (John is smiling), then the head verb is labeled as an AUXiliary, otherwise it is a Verb (John is happy).

3. **Copula vs. auxiliary.** A related problem is the distinction between v:cop and aux for the verb to be. This problem arises mostly when the verb is followed by the past participle, as in I was finished. For these constructions, we take the approach that the verb is always the copula, unless there is a by phrase marking the passive.

4. **Communicators.** COmmunicators can be hard to distinguish imperatives or locative adverbs, especially at the beginning of a sentence. Consider a sentence such as there you are where there could be interpreted as either specifying a location vs. there is a car in which there is pro:exist.

### 11.14 POSTMOD

This tool enables you to modify the Brill rules of a database. There are these options:

- **+dF** use POST database file F (default is eng.db).
- **+rF** specify name of file (F) containing actions that modify rules.
- **+c** force creation of Brill's rules.
- **+lm** reduce memory use (but increase processing time).
12 GRASP – Syntactic Dependency Analysis

This chapter, written by Eric Davis, Shuly Wintner, Brian MacWhinney, Alon Lavie, and Kenji Sagae, describes a system for coding syntactic dependencies in the English CHILDES corpora. This system uses the GRASP program in CLAN to process the information on the %mor line to automatically create a dependency analysis. Here we describe first the annotation system and then the use of the program.

12.1 Grammatical Relations

GRASP describes the structure of sentences in terms of pairwise grammatical relations between words. These grammatical relations involve two dimensions: attachment and valency. In terms of attachment, each pair has a head and a dependent. Along the valency dimension, each pair has a predicate and an argument. Each dependency relation is labeled with an arc and the arc has an arrow which points from the predicate to argument. Valency relations open up slots for arguments. In English, modifiers (adjectives, determiners, quantifiers) are predicates whose arguments are the following nouns. In this type of dependency organization the argument becomes the head. However, in other grammatical relations, the predicate or governor is the head and the resultant phrase takes on its functions from the predicate. Examples of predicate-head GRs include the attachment of thematic roles to verbs and the attachment of adjuncts to their heads. Here is an example of the coding of the sentence the big dog chased five cats for dependencies:

*TXT:   the big dog chased five cats.
%mor:   det|the adj|big n|dog v|chase-PAST quant|five n|cat
%gra:   1|3|DET 2|3|MOD 3|4|SUBJ 4|0|ROOT 5|6|QUANT 6|4|OBJ

This notation can be described in this way:

1. The determiner the is the first item and it attaches to the third item dog. Here the determiner is the predicate and the dependent. The GR here is DET or determination.
2. The adjective big is a predicate that attaches as a dependent of dog. The GR here is MOD or modification.
3. The noun dog is the head of the phrase the big dog and it attaches as a dependent subject or SUBJ of the predicate chased. Here we ignore the attachment of the suffix –ed to the verb.
4. The verb chased is the root of the clause. It attaches to the zero position which is the “root” of the sentence.
5. The quantifier five attaches to the noun cats through the QUANT relation.
6. The noun cats attaches as a dependent to the verb chased through the OBJ or object relation.

The following diagram describes the sentence We eat the cheese sandwich graphically
through arcs with arrowheads instead of through the numbering system:

```
*TXT:   we eat the cheese sandwich.
%mor:   pro|we v|eat det|the n|cheese n|sandwich .
%gra:    1|2|SUBJ 2|0|ROOT 3|5|DET 4|5|MOD 5|2|OBJ
```

This picture is equivalent to this notation in the numbering system:

The creation of these %gra notations, either by hand or by machine, depends on three levels of representation:

1. Words must be consistently coded for the correct part of part of speech on the %mor line. For English, the relevant categories are given in the MOR grammars. The major categories are adjective, adverb, pronoun, noun, and verb. However, there are many additional subcategories. Compounds are treated as single units given in their main part of speech by MOR.

2. GRASP makes use of a set of 42 grammatical relations (GRs) given below.

3. GRs apply to phrases or clusters of words connected through dependency relations. The most important phrasal types are: noun phrase, verb phrase, absolute phrase, gerund phrase, auxiliary phrase, and infinitival phrase.

The following is a comprehensive list of the grammatical relations in the GRASP annotation scheme. Example GRs as well as other relevant GR to that particular GR are provided. In this annotation scheme, C refers to clausal and X refers to non-finite clausal. This list is divided into relations with the predicate as head and relations with the argument as head. In the examples, the dependent is marked in italics.

**Predicate-head relations.** First, we list the relations in which the dependent attaches to a head that serves as the predicate. In many of these relations, the head is the verb. The combination of a verb with all of its arguments, including the SUBJ argument, constitutes a verb phrase.

1. **SUBJect** identifies the subject of clause, when the subject itself is not a clause. Typically, the head is the main verb and the dependent is a nominal. Ex: *You eat with your spoon.*

2. **ClausalSUBJect = CSUBJ** identifies the finite clausal subject of another clause.
The head is the main verb, and the dependent is the main verb of the clausal subject. An alternative analysis of these structures would treat the subordinator “that” as the head of the CSUBJ. Ex: That Eric cried moved Bush.

3. OBJect identifies the first object or direct object of a verb. The head is the main verb, and the dependent is a nominal or a noun that is the head of a nominal phrase. A clausal complement relation should be denoted by COMP or XCOMP (depending on whether the clausal complement is finite or non-finite, see below), not OBJ or OBJ2. Ex: You read the book.

4. OBJect2 = OBJ2 identifies the second object or indirect object of a ditransitive verb, when not introduced by a preposition. The head is a ditransitive verb, and the dependent is a noun (or other nominal). The dependent must be the head of a required non-clausal and nonprepositional complement of a verb (head of OBJ2) that is also the head of an OBJ relation. Ex: He gave you your telephone. When the indirect object is in a prepositional phrase, it is just coded as a prepositional phrase.

5. LOCative identifies the relation between a verb and a required location, as expressed by a PP or locative adverb. Locations are required for verbs such as put or live. They also can serve as the sole argument for the copulas be and become. Use LOC instead of JCT when the PP is required by the verb. Ex: Put the toys in the box. He lives there. He is in the hospital.

6. COMPlment identifies a finite clausal complement of a verb. The head is the main verb of the matrix clause, and the dependent is the main verb of the clausal complement. Ex: I think that was Fraser.

7. XCOMPlment identifies a non-finite clausal complement of a verb. The head is the main verb of the matrix clause, and the dependent is the main verb of the clausal complement. The XCOMP relation is only used for non-finite clausal complements, not predicate nominals or predicate adjectives (see PRED). Ex: You’re going to stand on my toe. I told you to go. Eve, you stop throwing the blocks.

8. PREDidate identifies a predicate nominal or predicate adjective of verbs such as be and become. The head is the verb. PRED should not be confused with XCOMP, which identifies a non-finite complement of a verb (some syntactic formalisms group PRED and XCOMP in a single category). Ex: I’m not sure. He is a doctor.

9. ClausalPREDidate = CPRED identifies a clausal predicate (finite or non-finite) that identifies the status of the subject of verbs such as be and become. The head is the main verb of the matrix clause, not its subject. The dependent is the verb of the predicate clause Ex: This is how I drink my coffee. My goal is to win the competition. (Maybe this should be changed to have the noun as the head?)

10. PrepositionalOBJect = POBJ is the relation between a preposition and its object. The head is a preposition, and the dependent is typically a noun. The traditional treatment of the prepositional phrase views the object of the preposition as the head of the prepositional phrase. However, we are here treating the preposition as the head, since the prepositional phrase then participates in a further JCT relation to a head verb or a NJCT relation to a head noun. Ex: You want to sit on the
11. SeRiaL identifies serial verbs such as go play and come see. In English, such verb sequences start with either come or go. The initial verb is the dependent, and the following verb is the head and typically the root of the sentence. Ex: *Come* see if we can find it. *Go* play with your toys over there. This relation can also be used when children overuse the pattern in forms with omitted infinitives such as *want see*, *try go*. It is not used with auxiliary *have*, although sometimes it seems that it could be.

**Argument-head relations:** Relations in which the arguments (rather than the predicates) serve as the heads include relations of adjunction and modification.

12. adJunCT = JCT identifies an adjunct that modifies a verb, adjective, or adverb. This grammatical relation covers a wide variety of structures whose exact form is best understood by noting the specific parts of speech that are involved. In all of these, the adjunct is the predicate, since it opens up a valency slot for something to attach to. The head of JCT is the verb, adjective or adverb to which the JCT attaches as a dependent. The dependent is typically an adverb or a preposition (in the case of phrasal adjuncts headed by a preposition, such as a prepositional phrase). Occasionally, a locative noun may function as an adjunct, as with *he was going all the way home*. Adjuncts are optional, and carry meaning on their own (and do not change the basic meaning of their JCT heads). Verbs requiring a complement describing location may be treated as prepositional objects, in which case the IOBJ relation applies (see above). Ex: That’s *much* better. He ran *with* a limp. That’s *really* big.

13. ClausaladJunCT = CJCT identifies a finite clause that adjoins to a verb, adjective, or adverb head. The dependent is typically the main verb of a subordinate clause and this clause attaches to the root verb of the main clause. The conjunction uses the LINK relation and attaches to the verb of the subordinate clause. Ex: We can’t find it, because it *is* gone.

14. XadJunCT = XJCT identifies a non-finite clause that attaches to a verb, adjective, or adverb. The dependent is typically the main verb of a non-finite subordinate clause. There is usually no conjunction for these. Ex: Running to the carriage, she lost her slipper. Note: this construction can be confused with ones in which a PP or adverb intervenes between an auxiliary and its participle, as in *she’s outside sleeping in the carriage*. However, when the main verb is a copula, as in *there’s a man sleeping in the car*, then XJCT is appropriate.

15. Nominal adJunCT = NJCT identifies the head of a complex NP with a prepositional phrase attached as an adjunct of a noun. Note that, if a prepositional phrase attaches to a verb, the relation is JCT and not NJCT. In a sense, this relation is a cross between JCT and MOD. Ex: The man *with* an umbrella arrived late.

16. MODifier identifies a non-clausal nominal modifier. This should not be confused with the part of speech code with the same name for modals. The link of modals to the main verb is coded as AUX. The head is a noun, and the dependent is typically an adjective or another noun, including a possessive. Ex: Would you like
grape juice? That’s a nice box. That’s John’s stick.

17. POSSessive is used for the relation between the English possessive suffix and the head noun. The head is the noun. The introduction of this relation is required in order to avoid problems in processing the verb clitics that take the same phonological shape. Ex: John’s book.

18. APPositive identifies the relation between an appositive phrase and its head. Ex: Barack Obama, President of the United States.

19. ClausalMODifier = CMOD identifies a finite clause that is a nominal modifier (such as a relative clause) or complement. The head is a noun, and the dependent is typically a finite verb. Ex: Here are the grapes I found.

20. XMODifier identifies a non-finite clause that is a nominal modifier (such as a relative clause) or complement. The head is a noun, and the dependent is typically a non-finite verb. Ex: It’s time to take a nap. I saw a man running away.

21. DETerminer identifies the relation between a determiner and its head noun. Determiners include the, a, as well as (adjectival) possessives pronouns (my, your, etc) and demonstratives (this, those, etc), but not quantifiers (all, some, any, etc; see QUANT below). Typically, the head is a noun and the dependent/governor is a determiner. In cases where a word that is usually a determiner does not have a head, there is no DET relation. Ex: I want that cookie.

22. QUANTifier identifies a nominal quantifier, such as three, many, and some. Typically, the head is a noun, and the dependent is a quantifier, or sometimes an adverb. In cases where a quantifier has no head, there is no QUANT relation. In English, the MOD, DET, and QUANT relations have largely the same syntax. However, within the noun phrase, we occasionally see that they are ordered as DET+QUANT+MOD+N. Ex: These are my three ripe bananas.

23. PostQuantifier = PQ is the relation between a postquantifier and the preceding head nominal. Ex: We both arrived late.

24. AUXiliary identifies an auxiliary or modal of a main verb. The head is a verb, and the dependent is an auxiliary (such as be or have) or a modal (such as can or should). Ex: Can you do it?

25. NEGation identifies verbal negation. When the word not (contracted or not) follows an auxiliary or modal (or sometimes a verb), it is the dependent in a NEG relation (not JCT), where the auxiliary, modal or verb (in the absence of an auxiliary or modal) is the head. Ex: Mommy will not read it.

26. INFinitive identifies the relation between the infinitival particle (to) and the verb to which it attaches. The head is a verb, and the dependent is always to. Ex: He’s going to drink the coffee.

27. LINK identifies the relation between a complementizer (that), relativizer (who, which) or subordinate conjunction (including and) and the verb in the subordinate clause to which it attaches. The verb of the subordinate clause attaches to the main verb in a CJCT, CMOD, CPRED, or COMP relation. Ex: Wait until the noodles are cool.

28. TAG is the relation between the finite verb of a tag question and the root verb of the main clause. Ex: You know how to count, don’t you? English and Portuguese have this structure, but many other languages do not.
Extra-clausal elements. In these relations, the dependent is a clausal modifier. These should be linked to the root.

29. COMmunicator identifies a communicator (such as *hey*, *okay*, etc) or a vocative. Because communicators and vocatives are typically global in a given sentence, the head of COM is “0”. Unlike BEG and END, COM items whose part of speech is “co” are not specifically marked with the ‡ on the main line, because they only have the “co” part of speech and so disambiguation in POST is never a problem for these items. However, this is not true for topics and vocatives. Those are marked as COM, but also need the ‡ mark for clear disambiguation.

30. BEG identifies an initial clause-external element, such as a vocative or topic. As with COM and TAG, the head of BEG is “0”. Ex: *Eve*, are you coming? BEG is marked in the main line with a following ‡ mark that is coded on the %mor line as beg‡.

31. END identifies a final clause-external postposed element. As with BEG, the head of the END is “0”. END is marked in the main line with a preceding „ mark that is coded on the %mor line as end„. Ex: Some more cookies, *Eve*? END differs from TAG in that it only contains one lexical items, whereas TAG has a core verb with additional arguments.

32. INCROOT identifies a word that serves as the root of an utterance, because the usual root forms (verbs in English) are missing. This form could be a single word by itself (adverb, communicator, noun, adjective) or a word with additional modifiers, such as the noun *dog* in *the big dog*, when it occurs by itself without a verb. It may appear that there could be more than one of these in an utterance, as in *well, sure*. However, in this case, *well* should be marked as a CO that is dependent on *sure*.

Cosmetic relations. There are several relations that are just used during transcription to assist in the accurate training of the GRASP tagger:

33. PUNCTuation is the relation between the final punctuation mark and the left wall or “0”.

34. LP (local punctuation) is the relation between quotes or commas and overall structure. Commas delimiting clauses should be linked to the root. Each comma in an ENUM series should be linked to the previous word. Beginning and ending quote marks should be linked to “0”.

35. BEGP is the relation between the ‡ mark and the BEG.

36. ENDP is the relation between the „ mark and the END.

37. ROOT This is the relation between the topmost word in a sentence (the root of the dependency tree) and the LeftWall or “0”. The topmost word in a sentence is the word that is the head of one or more relations, but is not the dependent in any relation with other words (except for the LeftWall).

Series relations. Some additional relations involve processes of listing, coordination, and classification. In these, the final element is the head and the initial elements all
depend on the final head. In English, this extends the idea that the last element in a compound is the head.

38. NAME identifies a string of proper names such as Eric Davis and New York Central Library. The initial name is the dependent, and the adjacent name is the head. The adjacent name is the dependent of the ROOT. Ex: My name is Tom Jones.

39. DATE identifies a date with month and year, month and day, or month, day, and year. Examples include: October 7, 1980 and July 26. For consistency, we regard the final element in these various forms as the head. Ex: October seventh nineteen ninety.

40. ENUMeration involves a relation between elements in a series without any coordination based on a conjunction (and, but, or). The series can contain letters, numbers, and nominals. The head is the first item in the series, and all the other items in the enumeration depend on this first word. Ex: one, two, three, four.

41. CONJ involves a relation between a coordinating conjunction and the preceding conjoined items, taken as a whole that has been combined with ENUM. For example, in the phrase I walk, jump, and run, the items walk and jump are combined by ENUM so that walk is the head. The conjunction and then attaches to walk with the CONJ relation. The resultant phrase walk, jump and is then further linked by the COORD relation to the final element ran. Ex: I walk, jump, and run.

42. COORD involves an attachment of a final coordinated element to the conjunction. For example, in the sentence I walk, jump, and run, the verb run is attached to and. Note: When the word and functions as a subordinating conjunction, it is treated as having a LINK relation to the verb of its clause. In this case, it is not conjoining phrases but whole clauses.

Bivalency. The above GRs describe dependencies in terms of the direction of the major valency relations. However, many of these relations have a secondary bivalent nature. For example, in the relations of thematic roles with the verb, it is also true that nominals are “looking for” roles in predications. Within the noun phrase, common nouns are looking for completion with either articles or plurality. Also, the attachment of auxiliaries to non-finite verbs serves to complete their finite marking. We can think of these additional valency relations as secondary relations. In all of these cases, valency works within the overall framework of dependency. Because GRASP relations are unidirectional, bivalency cannot be represented in GRASP.

12.2 Ellipsis and Omission

When ellipsis or omission leaves determiners or other modifiers without heads, then they should be attached to the most local predicate using the OM (Omission) relation.

12.3 GRs for Spanish

The GRs needed for Spanish are close to those for English. The distinction between X and C in the XJCT/CJCT and XMOD/CMOD relations are irrelevant to Spanish because
the finite/nonfinite status of the verb is clearly reflected in its morphology. So, the XJCT and XMOD relations can be dropped. The two additional relations that are needed are

1. **APREP** for the a-personal. This relation can either mark the direct object, when no OBJ is present or the indirect object when there is an OBJ relation. The APREP attaches to the verb and the object of the preposition attaches to the “a” with the POBJ relation.

2. **INF** for the preposition introducing an infinitive. For example, in *lo hizo para salir*, the preposition para relates to the infinitive *salir* with the INF relation. The infinitive then relates to the main verb with a COMP relation.

### 12.4 GRs for Chinese

The GRs needed for Chinese are not too very different from those needed for English. The major differences involve these areas:

1. Chinese uses all the basic GRs that are also found in English with these exceptions: TAG, DET, and INF.

2. Also, Chinese does not have a finite, non-finite distinction on verbs. Somewhat arbitrarily, this makes the “X” relations of English irrelevant and only CSUBJ, COMP, CMOD, CPRED, and CJCT are needed. Also, the main verb is the head of the clause, not the auxiliary.

3. Chinese makes extensive use of topics and sentence final particles. In the TOP relation, the head/root can be a nominal or adjective, as well as a verb.

4. Chinese has a variety of verb-verb constructions, beyond simple serial verbs. For Chinese, the head of SRL is the first verb, not the second.

5. The Chinese possessive construction has the head in final position. Also, Chinese can use classifier phrases such as *yi1 bian4* as JCT.

6. Chinese often combines clauses without using any conjunction to mark subordination or coordination. In this case, it is best to transcribe the two clauses as separate sentences. To mark the missing subordination relation, just add this postcode to the end of the first sentence: [+sub]. This mark does not necessarily imply which clause is subordinate; it just notes that the two clauses are related, although the relation is not marked with a conjunction.

7. The CJCT relation can also extend to *mei2 you3* because they are both words: neg|mei2=not v|you3=have v|jiang3=speak v:resc|wan2=finish.

8. The CJCT relation is also used in the many cases where there are no subordinating conjunctions, as in this example:

9. Clefts can be coded using the CPRED relations as in this example:

10. The JCT relation for Chinese extends also to the complements of directional verbs, as in this example:
The additional relations needed for Chinese are:

1. **POSSession = POSS** is the relation that holds between the linker “de” and the preceding possessor noun or pronoun which then functions as the head for further attachment to the thing possessed through the MOD relation.

2. Chinese has no articles and uses classifiers (CLASS) to mark quantification.

3. **VR** is the relation between the initial verb as head and the following resultative verb as dependent.

4. **VD** is the relations between the initial verb in a serial verb construction as head and the following directional verb as dependent.

5. **SFP** is a relation between the sentence final particle and “0”.

6. **PTOP** is a postposed topic, as in this example:
   
   `pro|zhe4=this class|ge4 v:cop|shi4=is pro:wh|shen2me=what prozlhe4=this classlge4 1|2|DET 2|3|SUBJ 3|0|ROOT 4|3|OBJ 6|3|SUBJ 6|2|SRL`

7. **PostpositionalObject = POSTO** is the relation between a postposition, which is the head and the dependent noun. This relation is marked on the noun, as in this example:
   
   `ta pao dao jiaoshu limian qu 1|2|SUBJ 2|0|ROOT 3|2|JCT 4|5|POSTO 5|3|JCT 6|2|SRL`

8. **PrepositionalObject = PREPO** is the relation between a preposition, which is the head and the following dependent noun, as in this example, where the relation is coded on the noun:
   
   `ta shi cong Beijing lai de 1|2|SUBJ 2|0|ROOT 3|5|JCT 4|3|PREPO 5|2|CJCT 6|5|LINK`
   
   This relation can also hold between a preposition and a postposition, as in this example:

   `v|yang3=maintain prep|zai4=at pro:wh|shen2me=what post|li3mian4=inside ? 1|0|ROOT 2|1|JCT 3|4|POSTO 4|2|PREPO 5|1|PUNCT`

### 12.1 GRs for Japanese

The GRs needed for Japanese are more differentiated than those needed for English. The major differences involve these areas:

1. Japanese is left-branched with the head following the modifier, adjunct or complement.

2. Japanese uses optional case particles to identify case relations. The head noun is coded as SUBJ or OBJ, resp. The case particles are coded as CASP, and constitute the head of the argument. Free adjuncts are coded as JCT, and the corresponding postparticles as POSTP.

   `Ken ga Tookyoo kara kita`
   
   Ken SUBJ Tokyo from come-PAST “Ken came from Tokyo”
   
   `1|2|SUBJ 2|5|CASP 3|4|JCT 4|5|POSTP 5|0|ROOT`

3. The root can be a tense-bearing element like a verb, a verbal adjective (ROOT), a copula (COPROOT) or a noun (PREDROOT).

4. The root can also be a case particle (CASPROOT), a postparticle (POSTPROOT), an
attributive particle (ATTPROOT), a quotative particle (QUOTPROOT), a topic particle (TOPPROOT), a quotative marker, a focus particle (FOCPROOT), or a conjunctive particle (CPZRROOT). Note that these different types of roots are fully grammatical and not elliptic fragments.

Papa no.
Dad GEN “(it's) Dad's one.”
1|2|MOD 2|0|ATTPROOT
iku kara.
go-PRES because “(I) will go”
1|2|COMP 2|0|CPZRROOT

5. Japanese expresses topic relations (TOP); the topic particle is coded as TOPP.
6. Like Chinese, Japanese has no articles and uses classifiers and counters to mark quantification.

sankurambo sanko tabeta.
cherry 3-pieces eat-PAST “he ate 3 cherries”
1|3|OBJ 2|1|QUANT 3|0|ROOT

7. Spoken Japanese makes extensive use of sentence final particles (SFP) and sentence modifiers (SMDR). They are depending of the preceding ROOT.

tabeta no ?
eat-PAST SFP “did you eat (it)?”
1|0|ROOT 2|1|SFP
tabeta jan.
eat-PAST SMDR “you ate it, didn't you?”
1|0|ROOT 2|1|SMDR

<table>
<thead>
<tr>
<th>Root</th>
<th>ROOT</th>
<th>COPROOT</th>
<th>PREDROOT</th>
<th>CPREDROOT</th>
<th>CASPROOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbal ROOT; relation between verb and left wall: v, adj, subsidiary verb (tense bearing element)</td>
<td>COPula ROOT; copula with noun, adjectival noun, or sentence nominalizer (no da)</td>
<td>nominal ROOT (without copula); includes adv, co, and quant, as well as verbal nouns and adjectival nouns in root position.</td>
<td>nominal ROOT with a sentence nominalizer in root position (ptl:snr=SNR)</td>
<td>CASe Particle ROOT (the case particle is the head)</td>
<td></td>
</tr>
<tr>
<td>CLAN Manual</td>
<td>208</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| TOPROOT | TOPic Particle ROOT  
\textit{kore wa?} ‘and what about this one?’  
\textit{112TOP 2\textcircled{0}TOPPROOT} |
| FOCROOT | FOCus Particle ROOT  
\textit{kore mo?} ‘this one, too?’  
\textit{112FOC 2\textcircled{0}FOCPROOT} |
| Topic | TOP  
‘TOPicization, (for convenience the root of the sentence is considered to be the head)  
\textit{kore wa yomenai}. ‘I can’t read it.’  
\textit{112TOP 2\textcircled{3}TOPP 3\textcircled{0}ROOT} |
| Topic | CTOP  
finite Clausal TOPic (head of the clause is ptl:snr/\textit{no})  
\textit{iku no wa ii kedo […]} ‘it’s ok to go, but…’  
\textit{112CMOD 2\textcircled{3}CTOP 3\textcircled{4}TOPP 4\textcircled{5}ICOMP 5\textcircled{6}CPZR} |
| FOC | FOCus (followed by ptl:foc; \textit{mo, shika, bakari, hodo etc.})  
\textit{kore mo yonda}. ‘he read this one, too.’  
\textit{112FOC 2\textcircled{3}FOCP 3\textcircled{0}ROOT} |
| Arguments | SUBJ  
nonclausal SUBject  
\textit{Jon ga tabeta}. ‘John ate it.’  
\textit{112SUBJ 2\textcircled{3}ICASP 3\textcircled{0}ROOT} |
| Arguments | CSUBJ  
finite Clausal SUBject (head of the clause is ptl:snr)  
\textit{taberu no ga ii}. ‘it’s good to eat it.’  
\textit{112CMOD 2\textcircled{3}CSUBJ 3\textcircled{4}ICASP 3\textcircled{0}ROOT} |
| Arguments | OBJ  
accusative OBJect  
\textit{hon o yonda}. ‘he has read the book.’  
\textit{112OBJ 2\textcircled{3}ICASP 3\textcircled{0}ROOT} |
| Arguments | COBJ  
finite Clausal accusative OBJect  
\textit{taberu no o yameta}. ‘he stopped eating.’  
\textit{112CMOD 2\textcircled{3}COBJ 3\textcircled{4}ICASP 4\textcircled{0}ROOT} |
| Adjuncts | JCT  
adJunCT (Postpositional or adverbial phrase)  
\textit{gakkoo kara kaetta}. ‘he came back from school.’  
\textit{112JCT 2\textcircled{3}POSTP 3\textcircled{0}ROOT} |
| Adjuncts | CJCT  
finite Clausal adJunCT  
\textit{ochita no de taberu}. ‘I’ll eat with the one that had fallen down.  
\textit{112CMOD 2\textcircled{3}CJCT 3\textcircled{4}POSTP 4\textcircled{0}ROOT} |
| Adjuncts | XJCT  
nonfinite clause as adJunCT (\textit{tabe-reba, -tara, -te, -cha, -tari; oishi-ku; shizuka ni})  
\textit{kaeseba ii}. ‘it’s ok to give it back.’  
\textit{112XJCT 2\textcircled{0}ROOT} |
| Clause conjunction | CPZR  
ComPlementiZeR (subordinating conjunctive particle; ptl:conj)  
\textit{osoi kara kaeru}. ‘I’ll go home because it’s late.’  
\textit{112ICOMP 2\textcircled{3}CPZR 3\textcircled{0}ROOT} |
<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCPZR</td>
<td>Zero-ComPlementiZeR (sentence introducing conjunction); head is always the root</td>
<td><em>dakara kaeru.</em> ‘that’s why I’ll go home.’</td>
<td>121ZCPZR 201ROOT</td>
</tr>
<tr>
<td>COMP</td>
<td>finite clausal verb COMplement (before ptl:conj and quotlto)</td>
<td><em>osoi kara kaeru.</em> ‘I’ll go home because it’s late.’</td>
<td>121COMP 231CPZR 301ROOT</td>
</tr>
<tr>
<td>QUOTP</td>
<td>QUOTation Particle after nominal or verbal phrase</td>
<td><em>kaeru to iimashita.</em> ‘he said he would go home.’</td>
<td>131COMP 211QUOTP 301ROOT</td>
</tr>
<tr>
<td>ZQUOT</td>
<td>Zero-QUOTative (sentence introducing quotative marker)</td>
<td><em>tte iu ka […]</em> ‘in other words’</td>
<td>121ZQUOT 2131COMP 341CPZR</td>
</tr>
</tbody>
</table>

**Nominal head**<br>**MOD** nonclausal MODifier (of a nominal) | *Papa no kutsu ga atta.* ‘there are Dad’s shoes.’ | 121MOD 2131CASP 3411SUBJ 4511CASP 301ROOT |
| CMOD      | finite Clausal MODifier of a nominal; the dependent is a finite verb, adjective or adj noun with copula | *akai kuruma o mita.* ‘he saw a red car.’ | 121CMOD 2131OBJ 3411CASP 401ROOT |
| XMOD      | nonfinite clausal MODifier of a nominal (adn) | *kore to onaji mono ga […]* ‘a thing similar to this one’ | 121JCT 2131POSTP 3411XMOD 4511SUBJ 5611CASP |
| COORD     | COORDination, second noun is the head; (ptl:cool) | *jyu to neko o katte iru.* ‘he has a dog and a cat.’ | 121COORD 2131COOP 3411OBJ 4511CASP 5611JCT 601ROOT |

**NP structure**<br>**PRED** nominal PREDicate before copula or QUOT | *tabeta hito da.* ‘he is the one who ate it.’ | 121CMOD 2131PRED 3011COPROOT |
<p>| CPRED     | finite Clausal PREDicate before copula (no da) | <em>taberu no da.</em> ‘in fact, he’ll eat it.’ | 121CMOD 2131CPRED 3011COPROOT |
| CASP      | CASE Particles (ptl:case; ga, o) | <em>hon o yonda.</em> ‘he read the book.’ | 121OBJ 2131CASP 301ROOT |
| POSTP     | POSTpositional Particles (ptl:post; ni, de, kara, made, to) | <em>Papa ni ageta.</em> ‘he gave it to Dad.’ | 121JCT 2131POSTP 301ROOT |
| ATTP      | ATTRibutive Particle | <em>Papa no kutsu (ga)</em> ‘Dad’s shoes are…’ | 121MOD 2131ATTP 341SUBJ |
| ATTP-SUBJ | ATTRibutive Particle in SUBJect position with head-noun elided | <em>Papa no ga atta.</em> ‘here is Dad’s one.’ | 121MOD 2131ATTP-SUBJ 3411CASP 401ROOT |
| ATTP-OBJ  | ATTRibutive Particle in OBJect position | <em>Papa no o mita.</em> ‘I saw Dad’s one.’ | 121MOD 2131ATTP-OBJ 3411CASP 401ROOT |</p>
<table>
<thead>
<tr>
<th>CLAN Manual</th>
<th>210</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SMDR</strong></td>
<td>sentence final Sentence ModifieR (smod</td>
</tr>
<tr>
<td><strong>SFP</strong></td>
<td>Sentence Final Particle (including the use after arguments and adjunct) <em>kuru ne</em>. ‘he’ll come, won’t he?’ 110</td>
</tr>
<tr>
<td><strong>COM</strong></td>
<td>COM municator; (co:il co:gl) including isolated final particles, sentence modalizers and onomatopoeias; head is always set to 0 <em>anoo tabeta</em>. ‘err..I ate it.’ 110</td>
</tr>
</tbody>
</table>
| **VOC** | **VOCAutive** ; head is always set to 0  
*Taishoo ‡ aka. ‘Taishoo, it’s red.’*
110|VOC 211|VOCP 30|PREDROOT |
|---|---|---|---|
| **Punctuation** | **PUNCT** | sentence boundary (sentence ends; .!? etc.); the root is the head  
*iku . ‘I’ll go.’*
110|ROOT 211|PUNCT |
| **RDP** | **Right Dislocation boundary** (dloc|=DISLOC); dislocation follows; the root is the head  
*mita ° fuusen ? ‘the balloon, did you see it?’*
110|ROOT 211|RDP 311|OBJ 411|PUNCT |
| **VOCP** | **VOCAtive marker** (voc|=VOC); head is the preceding vocative  
*Taishoo ‡ mite ! ‘Taishoo, look!’*
110|VOC 211|VOCP 30|ROOT 43|PUNCT |

### 12.2 MEGRASP

The MEGRASP program uses a megrasp.mod database file to create a dependency parsing based on binary grammatical relations. These relations are described earlier in this chapter. In its default mode, MEGRASP uses the megrasp.mod file and the part of speech tags on the %mor line to create a new %gra line that encodes these dependencies. To visualize the graph of these dependencies, you can triple click on a %gra line and CLAN will run a server-side web service program that draws out a graph structure to your screen.

To create a new megrasp.mod database file, you can run MEGRASP in training mode used the –t switch. In that case, your training corpus must contain a gold standard %grt tier to control training. The program uses these options:

- **-e**: evaluate accuracy (input file must contain gold standard GRs)
- **-t**: training mode (parser runs in parse mode by default)
- **-iN**: number of iterations for training ME model (default: 300)
- **-cN**: inequality parameter for training ME model (default: 0.5)
- **+fS**: send output to file (program will derive filename)
- **+re**: run program recursively on all sub-directories.
## 13 Utility Commands Table

The various utility commands are used primarily for fixing and reformatting older files to bring them into accord with the current CHAT format or for reformatting data for use with other programs.

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGLES</td>
<td>213</td>
<td>Enter coding lines for multiple video angles</td>
</tr>
<tr>
<td>ANVIL2CHAT</td>
<td>213</td>
<td>Convert ANVIL files to CHAT format</td>
</tr>
<tr>
<td>CHAT2ANVIL</td>
<td>213</td>
<td>Convert CHAT files to ANVIL format</td>
</tr>
<tr>
<td>CHAT2CA</td>
<td></td>
<td>Convert CA/CHAT to purer CA for display only</td>
</tr>
<tr>
<td>CHAT2CONNL</td>
<td>213</td>
<td>Convert to CONNL format</td>
</tr>
<tr>
<td>CHAT2ELAN</td>
<td>213</td>
<td>Convert CHAT to ELAN format</td>
</tr>
<tr>
<td>CHAT2PRAAT</td>
<td>214</td>
<td>Convert CHAT to Praat format</td>
</tr>
<tr>
<td>CHAT2XMAR</td>
<td>214</td>
<td>Convert CHAT to EXMARaLDA format</td>
</tr>
<tr>
<td>CHSTRING</td>
<td></td>
<td>Changes words and characters in CHAT files</td>
</tr>
<tr>
<td>CMDI</td>
<td>213</td>
<td>Create CMDI metadata files</td>
</tr>
<tr>
<td>COMBINE</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td>COMBTIER</td>
<td>217</td>
<td>Combines extra commentary lines</td>
</tr>
<tr>
<td>COMPOUND</td>
<td>216</td>
<td>Converts word pairs to compounds</td>
</tr>
<tr>
<td>CONNL2CHAT</td>
<td>216</td>
<td>Converts CONNL format to CHAT</td>
</tr>
<tr>
<td>CP2UTF</td>
<td>217</td>
<td>Converts ASCII files to Unicode files.</td>
</tr>
<tr>
<td>DATACLEAN</td>
<td>217</td>
<td>Updates the format of older CHAT files.</td>
</tr>
<tr>
<td>DATES</td>
<td>218</td>
<td>Uses the date and birthdate of the child to compute age.</td>
</tr>
<tr>
<td>DELIM</td>
<td>218</td>
<td>Inserts periods when final delimiters are missing.</td>
</tr>
<tr>
<td>ELAN2CHAT</td>
<td>218</td>
<td>Converts ELAN files to CHAT</td>
</tr>
<tr>
<td>FIXBULLETS</td>
<td>219</td>
<td>Repairs bullets and reformats old style bullets</td>
</tr>
<tr>
<td>FIXIT</td>
<td>219</td>
<td>Breaks up tiers with multiple utterances.</td>
</tr>
<tr>
<td>FIXLANG</td>
<td>219</td>
<td>--</td>
</tr>
<tr>
<td>FIXMP3S</td>
<td>219</td>
<td>Fixes bullets to MP3 media in older CHAT files</td>
</tr>
<tr>
<td>FLO</td>
<td>219</td>
<td>Reformats the file in simplified form.</td>
</tr>
<tr>
<td>INDENT</td>
<td>219</td>
<td>Aligns the overlap marks in CA files.</td>
</tr>
<tr>
<td>INSERT</td>
<td>220</td>
<td>Inserts @ID fields.</td>
</tr>
<tr>
<td>JOINITEMS</td>
<td>220</td>
<td>--</td>
</tr>
<tr>
<td>LAB2CHAT</td>
<td>220</td>
<td>Converts Wavesurfer format to CHAT</td>
</tr>
<tr>
<td>LIPP2CHAT</td>
<td>220</td>
<td>Converts LIPP files to CHAT</td>
</tr>
<tr>
<td>LONGTIER</td>
<td>220</td>
<td>Removes carriage returns to make long lines.</td>
</tr>
<tr>
<td>LOWCASE</td>
<td>220</td>
<td>Converts uppercase to lowercase throughout a file.</td>
</tr>
<tr>
<td>MAKEMOD</td>
<td>216</td>
<td>Create a %mod line using CMULEX</td>
</tr>
<tr>
<td>OLAC</td>
<td>222</td>
<td>Creates XML index files for the OLAC database</td>
</tr>
<tr>
<td>ORT</td>
<td>222</td>
<td>Converts Chinese characters</td>
</tr>
<tr>
<td>PRAAT2CHAT</td>
<td>222</td>
<td>Converts PRAAT files to CHAT</td>
</tr>
<tr>
<td>QUOTES</td>
<td>223</td>
<td>Moves quoted material to its own tier.</td>
</tr>
</tbody>
</table>
REPEAT 223 Inserts postcodes to mark repeated utterances.
RETRACE 223 Inserts retrace markers.
RTFIN 216 Converts RTF format to CHAT
SALTIN 223 Converts SALT files to CHAT format.
SILENCE 224 Converts utterances marked with keywords to silence
SPREADSHEET 216 Rotate table by turning columns into row and vice versa
SUBTITLES 216 Converts subtitled text files to CHAT text files
SYNCODING 224 --
TEXTIN 224 Converts straight text to CHAT format.
TIERORDER 225 Rearranges dependent tiers into a consistent order.
TRIM 225 Removes individual coding tiers.
TRNFIX 225 Compares the %trn and %mor lines.
UNIQ 225 Sorts lexicon files and removes duplicates.
USEDLEX 216 Creates a subset of lex files using only words in a corpus

### 13.1 ANGLES

This program is used to insert additional time marks for multiple video views or angles linked to a single transcript. The format of the file is illustrated in the angles.cha file in the /examples directory in the CLAN distribution.

### 13.2 ANVIL2CHAT

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

### 13.3 CHAT2ANVIL

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

### 13.4 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.

### 13.5 CHAT2CONLL

This program creates CONLL formatted files from the information on the CHAT %mor and %gra tiers, using the translation schemes provided in the 0pos.cut, 0mor.cut, and 0gr.cut files for parts of speech, morphological tags, and grammatical relations,
respectively.

13.6 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.

13.7 CHAT2PRAAT

This program converts a CHAT file to the Praat format.

13.8 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/.

13.9 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:

+"teddy\_bear" "teddy+bear" +"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 a number of CHSTRING clean-up actions. It turns off deletion of blank lines, removal of blank spaces, removal of empty dependent tiers, replacement of spaces after headers with a tab, and wrapping of long lines. All it allows is the replacement of individual strings.

+l Work only on material that is to the left of the colon which follows the tier ID. For example, if you want to add an “x” to the %syn to make it %xsyn, you would use this command:

chstring +s"%mor:" "%xmor:" +t% +l *.cha

+q CHAT requires that a three letter speaker code, such as *MOT:, be followed by a tab. Often, this space is filled by three spaces instead. Although this is undetectable visually, the computer recognizes tabs and spaces as separate entities. The +q option brings the file into conformance with CHAT by replacing the spaces with a tab. It also reorganizes lines to wrap systematically at 80 characters.

+s Sometimes you need to change just one word, or string, in a file(s). These strings can be put directly on the command line following the +s option. For example, if you wanted to mark all usages of the word gumma in a file as child-based forms, the option would look like this:

+s"gumma" "gumma@c"

+w Do string-oriented search and replacement, instead of word-oriented search and replacement. CAUTION: Used incorrectly, the +w switch can lead to serious losses of important data. Consider what happens when changing all occurrences of “yes” to “yeah.” If you use this command with the +w switch included,
chstring +w +s"yes" "yeah" myfile.cha

every single occurrence of the sequence of letters y-e-s will be changed. This includes words, such as “yesterday,” “eyes,” and “polyester,” which would become “yahterday,” “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

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

13.10 CMDI

This program runs over the CHILDES and TalkBank databases to produce a complete metadata inventory in CMDI format.

13.11 COMBINE

This program combines multiple files that coded participants separately into a single file based on the time codes.

13.12 COMPOUND

This program changes pairs of words to compounds, to guarantee more uniformity in morphological and lexical analysis. It requires that the user create a file of potential compound words in a format with each compound on a separate line, as in this example.

night+night
Chatty+baby
oh+boy

Whenever the program finds “night night” in the text, whether it be written as “night+night”, “night night” or “night-night,” it will be changed to “night+night”.
13.13 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.

13.14 CONLL2CHAT

This program takes a file in CONLL format and outputs a CHAT file with \%mor and \%syn lines, using the translations given in the 0pos.cut, 0mor.cut, and 0gr.cut files. The +c switch allows the user to state the customization required for particular flavors of the overall CoNLL format, as in +cpr for the PROIEL format.

13.15 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 the Library directory to effect translation of ASCII to Unicode for IPA symbols, depending on the nature of the ASCII IPA being used. For example, the +c3 switch produces a translation from IPAPhon. The +@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 ...)

13.16 DATACLEAN

DATACLEAN is used to rearrange and modify old style header tiers and line identifiers.
1. If @Languages tier is found, it is moved to the position right after @Begin.
2. If @Participants tier is found, it is moved to the position right after @Languages.
3. If the tier name has a space character after ':', then it is replaced with tab. If the tier name doesn't have a following tab, then it is added. If there is any character after the tab following a speaker name, such as another tab or space, then it is removed.
4. Tabs in the middle of tiers are replaced with spaces.
5. If utterance delimiters, such as +..., are not separated from the previous word with a space, then a space is inserted.
6. If [...] is not preceded or followed by space, then space is added.
7. Replaces #long with ###.
8. The string "..." is replaced with "+...".

13.17 DATES

The DATES program takes two time values and computes the third. It can take the child’s age and the current date and compute the child’s date of birth. It can take the date of birth and the current date to compute the child’s age. Or it can take the child’s age and the date of birth to compute the current date. For example, if you type:

```
dates +a 2;3.1 +b 12-jan-1962
```

you should get the following output:

```
@Age of Child: 2;3.1
@Birth of Child: 12-JAN-1962
@Date: 13-APR-1964
```

Unique Options

+a  Following this switch, after an intervening space, you can provide the child’s age in CHAT format.

+b  Following this switch, after an intervening space, you can provide the child’s birth date in day-month-year format.

+d  Following this switch, after an intervening space, you can provide the current date or the date of the file you are analyzing in day-month-year format.

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

13.18 DELIM

DELIM inserts a period at the end of every main line if it does not currently have one.

13.19 ELAN2CHAT

This program converts ELAN files to CHAT files. Use CHAT2ELAN for conversion in the opposite direction. The command is just: ELAN2CHAT filename.cha.
13.20 **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 +I switch can be used to make the implicit declaration of second language source explicit. The +o switch can be used to change the overall start time for a file.

- **+b** merge multiple bullets per line into one bullet per tier
- **+oN** time offset value N (+o+800 means add 800, +o-800 means subtract)
- **+fS** send output to file (program will derive filename)
- **-f** send output to the screen or pipe
- **+l** add language tag to every word
- **+re** run program recursively on all sub-directories.
- **+tS** include tier code S
- **-tS** exclude tier code S
- **+/-2** +2 do not create different versions of output file names / -2 create them

File names can be "+*:cha" or a file of list of names "+@:filename"

13.21 **FIXIT**

FIXIT is used to break up tiers with multiple utterances into standard format with one utterance per main line.

13.22 **FIXLANG**

Changes the codes for languages to the three-letter ISO standard.

13.23 **FIXMP3S**

This program fixes errors in the time alignments of sound bullets for MP3 files in versions of CLAN before 2006.

13.24 **FLO**

The FLO program creates a simplified version of a main CHAT line. This simplified version strips out markers of retracing, overlaps, errors, and all forms of main line coding. The only unique option in FLO is +d, which replaces the main line, instead of just adding a %flo tier.

FLO also uses several options that are shared with other commands. For a complete list of options for FLO, type the name of the command followed by a carriage return in the Commands window.

13.25 **INDENT**

This program is used to realign the overlap marks in CA files. The files must be in a
fixed width font such as FixedSysExcelsior.

**13.26 INSERT**

Programs such as MLU, and MLT can use the information contained in the @ID header to control their operation. After creating the @Participants line, you can run INSERT to automatically create @ID headers. After this is done, you may need to insert additional information in these header tiers by hand.

**13.27 JOINITEMS**

This program finds the items specified by the +s switch and joins them with the previous words. It is useful for reformatting Japanese particles as suffixes.

**13.28 LAB2CHAT**

This program converts files in WaveSurfer format to CHAT format.

**13.29 LIPP2CHAT**

In order to convert LIPP files to CHAT, you need to go through two steps. The first step is to run this command:

```
cp2utf -c8 *.lip
```

This will change the LIPP characters to CLAN’s UTF8 format. Next you run this command:

```
lipp2chat -len *.utf.cex
```

This will produce a set of *.utf.cha files which you can then rename to *.cha. The obligatory +l switch requires you to specify the language of the transcripts. For example “en” is for English and “fr” is for French.

**13.30 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, since it eliminates having to think about string replacements across line breaks.

**13.31 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.
13.32 MAKEMOD

This program uses the CMU Pronouncing Dictionary to insert phonological forms in SAMPA notation. The dictionary is copyrighted by Carnegie Mellon University and can be retrieved from http://www.speech.cs.cmu.edu/cgi-bin/cmudict/ Use of this dictionary, for any research or commercial purpose, is completely unrestricted. For MAKEMOD, we have created a reformatting of the CMU Pronouncing Dictionary that uses SAMPA. In order to run MAKEMOD, you must first retrieve the large cmulex.cut file from the server and put it in your library directory. Then you just run the program by typing

\texttt{makemod +t* filename}

By default, the program analyzes all speakers. However, you can control this using the +t switch. Also, by default, it only inserts the first of the alternative pronunciations from the CMU Dictionary. However, you can use the +a switch to force inclusion of all of them.

The beginning of the cmulex.cut file gives the acknowledgements for construction of the file and the following set of

<table>
<thead>
<tr>
<th>CMU</th>
<th>Word</th>
<th>full CMU</th>
<th>SAMPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>odd</td>
<td>AAD</td>
<td>A</td>
</tr>
<tr>
<td>AE</td>
<td>at</td>
<td>AE T</td>
<td>{</td>
</tr>
<tr>
<td>AH</td>
<td>hut</td>
<td>HH AH T</td>
<td>@</td>
</tr>
<tr>
<td>AO</td>
<td>ought</td>
<td>AO T</td>
<td>O</td>
</tr>
<tr>
<td>AW</td>
<td>cow</td>
<td>K AW</td>
<td>aU</td>
</tr>
<tr>
<td>AY</td>
<td>hide</td>
<td>HH AY D</td>
<td>aI</td>
</tr>
<tr>
<td>B</td>
<td>be</td>
<td>B IY</td>
<td>b</td>
</tr>
<tr>
<td>CH</td>
<td>cheese</td>
<td>CH IY Z</td>
<td>tS</td>
</tr>
<tr>
<td>D</td>
<td>dee</td>
<td>D IY</td>
<td>d</td>
</tr>
<tr>
<td>DH</td>
<td>thee</td>
<td>DH IY</td>
<td>D</td>
</tr>
<tr>
<td>EH</td>
<td>Ed</td>
<td>EH D</td>
<td>E</td>
</tr>
<tr>
<td>ER</td>
<td>hurt</td>
<td>HH ER T</td>
<td>3</td>
</tr>
<tr>
<td>EY</td>
<td>ate</td>
<td>EY T</td>
<td>eI</td>
</tr>
<tr>
<td>F</td>
<td>fee</td>
<td>F IY</td>
<td>f</td>
</tr>
<tr>
<td>G</td>
<td>green</td>
<td>G R IY N</td>
<td>g</td>
</tr>
<tr>
<td>HH</td>
<td>he</td>
<td>HH IY</td>
<td>h</td>
</tr>
<tr>
<td>IH</td>
<td>it</td>
<td>IH T</td>
<td>I</td>
</tr>
<tr>
<td>IY</td>
<td>eat</td>
<td>IY T</td>
<td>i</td>
</tr>
<tr>
<td>JH</td>
<td>gee</td>
<td>JH IY</td>
<td>dZ</td>
</tr>
<tr>
<td>K</td>
<td>key</td>
<td>K IY</td>
<td>k</td>
</tr>
<tr>
<td>L</td>
<td>lee</td>
<td>L IY</td>
<td>l</td>
</tr>
<tr>
<td>M</td>
<td>me</td>
<td>M IY</td>
<td>m</td>
</tr>
<tr>
<td>N</td>
<td>knee</td>
<td>N IY</td>
<td>n</td>
</tr>
<tr>
<td>NG</td>
<td>ping</td>
<td>P IH NG</td>
<td>N</td>
</tr>
<tr>
<td>OW</td>
<td>oat</td>
<td>OW T</td>
<td>o</td>
</tr>
<tr>
<td>OY</td>
<td>toy</td>
<td>T OY</td>
<td>OI</td>
</tr>
<tr>
<td>P</td>
<td>pee</td>
<td>P IY</td>
<td>p</td>
</tr>
</tbody>
</table>
The CMU Pronouncing Dictionary tone markings were converted to SAMPA in the following way. L0 stress was unmarked. A double quote preceding the syllable marked L1 stress and a percentage sign preceding the syllable being stressed marked L2 stress.

### 13.33 OLAC

This program goes through the various directories in CHILDES and TalkBank and creates an XML database that can be used by the OLAC (Online Language Archives Community) system to help researchers locate corpora relevant to their research interests.

### 13.34 ORT

ORT, if +c used, then this converts HKU style disambiguated pinyin with capital letters to CMU style lowercase pinyin on the main line. Without the +c switch, it is used to create a %ort line with Hanzi characters corresponding to the pinyin-style words found on main line. The choice of characters to be inserted is determined by entries in the lexicon files at the end of each word's line after the '%' character.

### 13.35 PRAAT2CHAT

This program converts files in the PRAAT format to files in CHAT format. You will find a folder in the CLAN /lib directory called praat2chat. This folder contains a readme.txt file that explains how to use the program. You need to set up an attribs.cut file to match what you have in your Praat file. However, when converting from a Praat file that was originally created in CLAN, no attribs.cut file is needed. To convert the e2at01f321.textGrid file in the sample directory to CHAT, use this command:

```
praat2chat -b -de2at01f_attribs.cut -opcl e2at01f321.textGrid
```

The attribs.cut file tells the program how to map from Praat dependent tier tags to CHAT tags. Take a look at the examples to see how this works.

After running praat2chat, you need to fix the headers, add zho to allow for numbers at the ends of words, and run DELIM to add periods for utterances. In general, you will need various similar cleanup operations for different corpora coming from Praat.
For converting from CHAT to Praat, just use this command:

```plaintext
  chat2praat +emp3 e2at01f321.cha
```

When going from a Praat file that was created with the chat2praat program, you do not need an attribs.cut file, because the tags will already be correctly formatted.

### 13.36 Quotes

This program moves quoted material to its own separate tier.

### 13.37 Repeat

Repeat if two consecutive main tiers are identical then the postcode [+ rep] is inserted at the end of the second tier.

### 13.38 Retrace

RETRACE inserts [/] after repeated words as in this example:

```plaintext
  *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.

### 13.39 RTFIN

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

### 13.40 SALTIN

This program takes SALT formatted files and converts them to the CHAT format. SALT is a transcript format developed by Jon Miller and Robin Chapman at the University of Wisconsin. By default, SALTIN sends its output to a file. Here is the most common use of this program:

```plaintext
  saltin file.cut
```

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

1. When SALTIN encounters material in parentheses, it translates this material as an unspecified retracing type, using the [/?] code.
2. Multiple comments are placed on separate lines on a single comment tier.
3. SALT codes in square brackets are converted to CHAT comments in square brackets and left in the original place they occurred, unless the + symbol is added to the SALT code. If it is present, the code is treated as a postcode and moved to the end of the utterance when SALTIN runs. The CHSTRING program can be used to insert + in the desired codes or in all codes, if required.
4. Times in minutes and seconds are converted to times in hours:minutes:seconds.
5. A %def tier is created for coding definitions.

#### Unique Options

- **+h** Some researchers have used angle brackets in SALT to enter comments. When
the original contains text found between the < and the > characters this option instructs SALTIN to place it between [% and ]. Otherwise, material in angle brackets would be coded as a text overlap.

+I Put all codes on a separate %cod line. If you do not select this option, codes will be placed on the main line in the [$text] format.

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

13.41 SILENCE

This program looks for utterances with some specific string and converts the linked segment of the related audio file to silence to preserve anonymity. By convention, the strings Lastname and Address are used to mark sentences to be silenced. However, other strings or postcodes can be used with the +s switch to the same effect.

13.42 SPREADSHEET

This program rotates a table by turning columns into rows and vice versa, or it can merge rows with matching cells into one row.

13.43 SUBTITLES

This program converts subtitle files into CHAT format.

13.44 SYNCODING

This program is not functional yet.

13.45 TEXTIN

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

```
@Begin
@Participants: T Text
*T:  the textin program is quite simple.
*T:  it takes a set of sentences in paragraph form and converts them to a chat file.
*T:  blank lines are considered to be possible paragraph breaks and are noted with @blank headers.
@End
```

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. There are no options that are unique to textin. However, it uses several options that are shared with
other commands. For a complete list of options for a command, type the name of the command followed by a carriage return in the Commands window. Information regarding the additional options shared across commands can be found in the chapter on Options.

13.46 TIERORDER

TIERORDER puts the dependent tiers into a consistent alphabetical order.

13.47 TRIM

This program is designed to allow you to remove particular 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
```

13.48 TRNFIX

This program compares the %trn line with the %mor line and notes discrepancies. This is useful in developing a training corpus for POSTTRAIN.

13.49 UNIQ

UNIQ is used to sort lexicon files into alphabetical order, while removing duplicates.

13.50 USEDLEX

This program removes all of the items in the lexicon files of a MOR grammar that are not being used in a given corpus. This is helpful for work with transcriptions that use a limited target vocabulary.
14 References


