Tools for Analyzing Talk

Part 3: Morphosyntactic Analysis

Brian MacWhinney
Carnegie Mellon University

May 24, 2023

https://doi.org/10.21415/T5B97X

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

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

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

1 Introduction

This third volume of the TalkBank manuals deals with the use of the programs that perform automatic computation of the morphosyntactic structure of transcripts in CHAT format. These manuals, the programs, and the TalkBank datasets can all be downloaded freely from https://talkbank.org.

The first volume of the TalkBank manual describes the CHAT transcription format. The second volume describes the use of the CLAN data analysis programs. This third manual describes the use of the MOR, POST, POSTMORTEM, and MEGRASP programs to add a %mor and %gra line to CHAT transcripts. The %mor line provides a complete part-of-speech tagging for every word indicated on the main line of the transcript. The %gra line provides a further analysis of the grammatical dependencies between items in the %mor line. These programs for morphosyntactic analysis are all built into CLAN.

Users who do not wish to create or process information on the %mor and %gra lines will not need to read this current manual. However, researchers and clinicians interested in these features will need to know the basics of the use of these programs, as described in the next chapter. The additional sections of this manual are directed to researchers who wish to extend or improve the coverage of MOR and GRASP grammars or who wish to build such grammars for languages that are not yet covered.

2 Morphosyntactic Coding

Linguists and psycholinguists rely on the analysis of morphosyntax to illuminate 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 in this tradition, 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.

The study of morphosyntax also plays an important role in the study and treatment of language disorders, such as aphasia, specific language impairment, stuttering, and dementia. For this work, both researchers and clinicians can benefit from methods for achieving accurate automatic analysis of correct and incorrect uses of morphosyntactic devices. To address these needs, the TalkBank system uses the MOR command to automatically generate candidate morphological analyses on the %mor tier, the POST command to disambiguate these analyses, and the MEGRASP command to compute grammatical dependencies on the %gra tier.

2.1 One-to-one correspondence

MOR creates a %mor tier with a one-to-one correspondence between words on the main line and words on the %mor tier. In order to achieve this one-to-one correspondence, the following rules are observed:

1. Each word group (see below) on the %mor line is surrounded by spaces or an initial tab to correspond to the corresponding space-delimited word group on the main line. The correspondence matches each %mor word (morphological word) to a main line word in a left-to-right linear order in the utterance.
2. Utterance delimiters are preserved on the %mor line to facilitate readability and analysis. These delimiters should be the same as the ones used on the main line.
3. Along with utterance delimiters, the satellite markers of ‡ for the vocative and „ for tag questions or dislocations are also included on the %mor line in a one-to-one

alignment format.

4. Retracings and repetitions are excluded from this one-to-one mapping, as are nonwords such as xxx or strings beginning with &. When word repetitions are marked in the form word [x 3], the material in parentheses is stripped off and the word is considered as a single form.
5. When a replacing form is indicated on the main line with the form [: text], the material on the %mor line corresponds to the replacing material in the square brackets, not the material that is being replaced. For example, if the main line has **gonna [: going to],** the %mor line will code **going to.**
6. The [*] symbol that is used on the main line to indicate errors is not duplicated on the %mor line.

2.2 Tag Groups and Word Groups

On the %mor line, alternative taggings of a given word are clustered together in *tag groups*. These groups include the alternative taggings of a word that are produced by the MOR program. Alternatives are separated by the ^ character. Here is an example of a tag group for one of the most ambiguous words in English:

```
adv|back^adj|back^n|back^v|back
```

After you run the POST program on your files, all of these alternatives will be disambiguated and each word will have only one alternative. You can also use the hand disambiguation method built into the CLAN editor to disambiguate each tag group case by case.

The next level of organization for the MOR line is the word group. Word groups are combinations marked by the preclitic delimiter $, the postclitic delimiter ~ or the compound delimiter +. For example, the Spanish word *dámelo* can be represented as

```
vimpsh|da-2S&IMP~pro:clit|1S~pro:clit|OBJ&MASC=give
```

This word group is a series of three words (verb~postclitic~postclitic) combined by the ~ marker. Clitics may be either preclitics or postclitics. Separable prefixes of the type found in German or Hungarian and other discontinuous morphemes can be represented as word groups using the preclitic delimiter $, as in this example for *ausgegangen* (“gone”):

```
prep|aus$PART#v|geh&PAST:PART=go
```

Note the difference between the coding of the preclitic “aus” and the prefix “ge” in this example. Compounds are also represented as combinations, as in this analysis of angelfish.

```
n|+n|angel+n|fish
```

Here, the first characters (n|) represent the part of speech of the whole compound and the subsequent tags, after each plus sign, are for the parts of speech of the components of the compound. Proper nouns are not treated as compounds. Therefore, they take forms with underlines instead of pluses, such as Luke_Skywalker or New_York_City.

2.3 Words

Beneath the level of the word group is the level of the word. The structure of each individual word is:

```
prefix#
part-of-speech|
stem
&fusionalsuffix
-suffix
=english (optional, underscore joins words)
```

There can be any number of prefixes, fusional suffixes, and suffixes, but there should be only one stem. Prefixes and suffixes should be given in the order in which they occur in the word. Since fusional suffixes are fused parts of the stem, their order is indeterminate. The English translation of the stem is not a part of the morphology, but is included for convenience for non-native speakers. If the English translation requires two words, these words should be joined by an underscore as in "lose_flowers" for French *défleurir*.

Now let us look in greater detail at the nature of each of these types of coding. Throughout this discussion, bear in mind that all coding is done on a word-by-word basis, where words are considered to be strings separated by spaces.

2.4 Part of Speech Codes

The morphological codes on the %mor line begin with a part-of-speech code. The basic scheme for the part-of-speech code is:

```
category:subcategory:subcategory
```

Additional fields can be added, using the colon character as the field separator. The subcategory fields contain information about syntactic features of the word that are not marked overtly. For example, you may wish to code the fact that Italian "andare" is an intransitive verb even though there is no single morpheme that signals intransitivity. You can do this by using the part-of-speech code **v:intrans**, rather than by inserting a separate morpheme.

In order to avoid redundancy, information that is marked by a prefix or suffix is not incorporated into the part-of-speech code, as this information will be found to the right of the **|** delimiter. These codes can be given in either uppercase, as in **ADJ**, or lowercase, as in **adj**. In general, CHAT codes are not case-sensitive.

The particular codes given below are the ones that MOR uses for automatic morphological tagging of English. Individual researchers will need to define a system of part-of-speech codes that correctly reflects their own research interests and theoretical commitments. Languages that are typologically quite different from English may have to use very different part-of-speech categories. Quirk, Greenbaum, Leech, and Svartvik (1985) explain some of the intricacies of part-of-speech coding. Their analysis should be taken as definitive for all part-of-speech coding for English. However, for many purposes, a more coarse-grained coding can be used.

The following set of top-level part-of-speech codes is the one used by the MOR program. Additional refinements to this system can be found by studying the organization of the lexicon files for that program For example, compounds use the main part-of-speech code, along with codes for their components. Further distinctions can be found by looking at the MOR lexicon.

English Parts of Speech

Category	Code
Adjective	adj
Adjective - Predicative	adj:pred
Adverb	adv
Adverb - Temporal	adv:tem
Communicator	co
Complementizer	comp
Conjunction	conj
Coordinator	coord
Determiner Article	det:art
Determiner - Demonstrative	det:dem
Determiner - Interrogative	det:int
Determiner - Numeral	det:num
Determiner - Possessive	det:poss
Filler	fil
Infinitive	inf
Negative	neg
Noun	n
Noun - letter	n:let
Noun - plurale tantum	n:pt
Proper Noun	n:prop
Onomatopoeia	on
Particle	part
Postmodifier	post
Preposition	prep
Pronoun - demonstrative	pro:dem
Pronoun - existential	pro:exist
Pronoun - indefinite	pro:indef
Pronoun - interrogative	pro:int
Pronoun - object	pro:obj
Pronoun - personal	pro:per
Pronoun - possessive	pro:poss
Pronoun - reflexive	pro:refl
Pronoun - relative	pro:rel
Pronoun - subject	pro:sub
Quantifier	qn
Verb	v
Verb - auxiliary	aux
Verb - copula	cop
Verb - modal	mod

2.5 Stems

Every word on the %mor tier must include a "lemma" or stem as part of the morpheme analysis. The stem is found on the right hand side of the | delimiter, following any pre-clitics or prefixes. If the transcript is in English, this can be simply the canonical form of the word. For nouns, this is the singular. For verbs, it is the infinitive. If the transcript is in another language, it can be the English translation. A single form should be selected for each stem. Thus, the English indefinite article is coded as **det|a** with the lemma "a" whether or not the actual form of the article is "a" or "an."

When English is not the main language of the transcript, the transcriber must decide whether to use English stems. Using English stems has the advantage that it makes the corpus more available to English-reading researchers. To show how this is done, take the German phrase "wir essen":

```
*FRI:   wir essen.
%mor:   pro|wir=we v|ess-INF=eat .
```

Some projects may have reasons to avoid using English stems, even as translations. In this example, "essen" would be simply **v|ess-INF**. Other projects may wish to use only English stems and no target-language stems. Sometimes there are multiple possible translations into English. For example, German "Sie"/sie" could be either "you," "she," or "they." Choosing a single English meaning for the stem helps fix the German form.

2.6 Affixes

Affixes and clitics are coded in the position in which they occur with relation to the stem. The morphological status of the affix should be identified by the following markers or delimiters: - for a suffix, # for a prefix, and **&** for fusional or infixed morphology.

The **&** is used to mark affixes that are not realized in a clearly isolable phonological shape. For example, the form "men" cannot be broken down into a part corresponding to the stem "man" and a part corresponding to the plural marker, because one cannot say that the vowel "e" marks the plural. For this reason, the word is coded as **n|man&PL**. The past forms of irregular verbs may undergo similar ablaut processes, as in "came," which is coded **v|come&PAST**, or they may undergo no phonological change at all, as in "hit", which is coded **v|hit&PAST.** Sometimes there may be several codes indicated with the **&** after the stem. For example, the form "was" is coded **v|be&PAST&13s**. Affix and clitic codes are based either on Latin forms for grammatical function or English words corresponding to particular closed-class items. MOR uses the following set of affix codes for automatic morphological tagging of English.

Inflectional Affixes for English

Function	Code
adjective suffix *er, r*	CP
adjective suffix *est, st*	SP
noun suffix *ie*	DIM
noun suffix *s, es*	PL

noun suffix *'s, '*	POSS
verb suffix *s, es*	3S
verb suffix *ed, d*	PAST
verb suffix *ing*	PRESP
verb suffix *en*	PASTP

Derivational Affixes for English

Function	Code
adjective and verb prefix *un*	UN
adverbializer *ly*	LY
nominalizer *er*	ER
noun prefix *ex*	EX
verb prefix *dis*	DIS
verb prefix *mis*	MIS
verb prefix *out*	OUT
verb prefix *over*	OVER
verb prefix *pre*	PRE
verb prefix *pro*	PRO
verb prefix *re*	RE

2.7 Clitics

Clitics are marked by a tilde, as in **v|parl&IMP:2S=speak~pro|DAT:MASC:SG** for Italian "parlagli" and **pro|it~v|be&3s** for English "it's." Note that part of speech coding with the **|** symbol is repeated for clitics after the tilde. Both clitics and contracted elements are coded with the tilde. The use of the tilde for contracted elements extends to forms like "sul" in Italian, "ins" in German, or "rajta" in Hungarian in which prepositions are merged with articles or pronouns.

Clitic Codes for English

Clitic	Code
noun phrase post-clitic *'d*	v:aux|would, v|have&PAST
noun phrase post-clitic *'ll*	v:aux|will
noun phrase post-clitic *'m*	v|be&1S, v:aux|be&1S
noun phrase post-clitic *'re*	v|be&PRES, v:aux|be&PRES
noun phrase post-clitic *'s*	v|be&3S, v:aux|be&3S
verbal post-clitic *n't*	neg|not

2.8 Compounds

Here are some words that we might want to treat as compounds: *sweatshirt*, *highschool*, *playground*, and *horseback*. You can find hundreds of these in files in the English lexicon such as adv+n+adv.cut that include the plus symbol in their names. There

are also many idiomatic phrases that could be best analyzed as linkages. Here are some examples: *a_lot_of*, *all_of_a_sudden*, *at_last*, *for_sure*, *kind_of*, *of_course*, *once_and_for_all*, *once_upon_a_time*, *so_far*, and *lots_of*. You can find hundreds of these in files in the English lexicon with names such as adj_under.cut.

On the %mor tier it is necessary to assign a part-of-speech label to each segment of the compound. For example, the word *blackboard* is coded on the %mor tier as **n|+adj|black+n|board**. The part of speech of the compound as a whole is usually given by the part-of-speech of the final segment, although this is not always true.

In order to preserve the one-to-one correspondence between words on the main line and words on the %mor tier, words that are not marked as compounds on the main line should not be coded as compounds on the %mor tier. For example, if the words "come here" are used as a rote form, then they should be written as "come_here" on the main tier. On the %mor tier this will be coded as **v|come_here**. It makes no sense to code this as **v|come+adv|here**, because that analysis would contradict the claim that this pair functions as a single unit. It is sometimes difficult to assign a part-of-speech code to a morpheme. In the usual case, the part-of-speech code should be chosen from the same set of codes used to label single words of the language. For example, some of these idiomatic phrases can be coded as compounds on the %mor line.

Phrases Coded as Linkages

Phrase	Phrase
qn|a_lot_of	adv|all_of_a_sudden
co|for_sure	adv:int|kind_of
adv|once_and_for_all	adv|once_upon_a_time
adv|so_far	qn|lots_of.

2.9 Punctuation Marks

MOR can be configured to recognize certain punctuation marks as whole word characters. In particular, the file punct.cut contains these entries:

```
„       {[scat end]} "end"
‡       {[scat beg]} "beg"
,       {[scat cm]} "cm"
```

When the punctuation marks on the left occur in text, they are treated as separate lexical items and are mapped to forms such as beg|beg on the %mor tier. The "end" marker is used to mark postposed forms such as tags and sentence final particles. The "beg" marker is used to mark preposed forms such as vocatives and communicators. These special characters are important for correctly structuring the dependency relations for the GRASP program.

2.10 Sample Morphological Tagging for English

The following table describes and illustrates a more detailed set of word class codings for English. The %mor tier examples correspond to the labellings MOR produces for the words in question. It is possible to augment or simplify this set, either by creating additional word categories, or by adding additional fields to the part-of-speech label, as

discussed previously. The entries in this table and elsewhere in this manual can always be double-checked against the current version of the MOR grammar by typing "mor +xi" to bring up interactive MOR and then entering the word to be analyzed.

Word Classes for English

Class	Examples	Coding of Examples
adjective	big	adj|big
adjective, comparative	bigger, better	adj|big-CP, adj|good&CP
adjective, superlative	biggest, best	adj|big-SP, adj|good&SP
adverb	well	adv|well
adverb, ending in ly	quickly	adv:adj|quick-LY
adverb, intensifying	very, rather	adv:int|very, adv:int|rather
adverb, post-qualifying	enough, indeed	adv|enough, adv|indeed
adverb, locative	here, then	adv:loc|here, adv:tem|then
communicator	aha	co|aha
conjunction, coord.	and, or	conj:coo|and, conj:coo|or
conjunction, subord.	if, although	conj:sub|if, conj:sub|although
determiner, singular	a, the, this	det|a, det|this
determiner, plural	these, those	det|these, det|those
determiner, possessive	my, your, her	det:poss|my
infinitive marker	to	inf|to
noun, common	cat, coffee	n|cat, n|coffee
noun, plural	cats	n|cat-PL
noun, possessive	cat's	n|cat~poss|s
noun, plural possessive	cats'	n|cat-PL~poss|s
noun, proper	Mary	n:prop|Mary
noun, proper, plural	Mary-s	n:prop|Mary-PL
noun, proper, possessive	Mary's	n:prop|Mary~poss|s
noun, proper, pl. poss.	Marys'	n:prop|Mary-PL~poss|s
noun, adverbial	home, west	n|home, adv:loc |home
number, cardinal	two	det:num|two
number, ordinal	second	adj|second
postquantifier	all, both	post|all, post|both
preposition	in	prep|in, adv:loc|in
pronoun, personal	I, me, we, us, he	pro|I, pro|me, pro|we, pro|us
pronoun, reflexive	myself, ourselves	pro:refl|myself
pronoun, possessive	mine, yours, his	pro:poss|mine, pro:poss:det|his
pronoun, demonstrative	that, this, these	pro:dem|that
pronoun, indefinite	everybody, nothing	pro:indef|everybody
pronoun, indef., poss.	everybody's	pro:indef|everybody~poss|s
quantifier	half, all	qn|half, qn|all
verb, base form	walk, run	v|walk, v|run
verb, 3rd singular present	walks, runs	v|walk-3S, v|run-3S

verb, past tense	walked, ran	v|walk-PAST, v|run&PAST
verb, present participle	walking, running	part|walk-PRESP, part|run-PRESP
verb, past participle	walked, run	part|walk-PASTP, part|run&PASTP
verb, modal auxiliary	can, could, must	aux|can, aux|could, aux|must

Since it is sometimes difficult to decide what part of speech a word belongs to, we offer the following overview of the major part-of-speech labels used in the standard English grammar.

ADJectives modify nouns, either prenominally, or predicatively. Unitary compound modifiers such as **good-looking** should be labeled as adjectives.

ADVerbs cover a heterogenous class of words including: manner adverbs, which generally end in **-ly**; locative adverbs, which include expressions of time and place; intensifiers that modify adjectives; and post-head modifiers, such as **indeed** and **enough.**

COmmunicators are used for interactive and communicative forms which fulfill a variety of functions in speech and conversation. Also included in this category are words used to express emotion, as well as imitative and onomatopeic forms, such as **ah**, **aw**, **boom**, **boom-boom**, **icky**, **wow**, **yuck**, and **yummy.**

CONJunctions conjoin two or more words, phrases, or sentences. Examples include: **although**, **because**, **if**, **unless,** and **until.**

COORDinators include **and, or,** and **as well as**. These can combine clauses, phrases, or words.

DETerminers include articles, and definite and indefinite determiners. Possessive determiners such as **my** and **your** are tagged **det:poss.**

INFinitive is the word "to" which is tagged **inf|to.**

INTerjections are similar to communicators, but they typically can stand alone as complete utterances or fragments, rather than being integrated as parts of the utterances. They include forms such as **wow, hello, good-morning, good-bye, please, thank-you**.

Nouns are tagged with **n** for common nouns, and n**:prop** for proper nouns (names of people, places, fictional characters, brand-name products).

NEGative is the word "not" which is tagged **neg|not.**

NUMbers are labelled **num** for cardinal numbers. The ordinal numbers are adjectives.

Onomatopoeia are words that imitate the sounds of nature, animals, and other noises.

Particles are words that are often also prepositions, but are serving as verbal particles.

PREPositions are the heads of prepositional phrases. When a preposition is not a part of a phrase, it should be coded as a particle or an adverb.

PROnouns include a variety of structures, such as reflexives, possessives, personal pronouns, deictic pronouns, etc.

QUANTifiers include **each, every, all, some,** and similar items.

Verbs can be either main verbs, copulas, or auxililaries.

2.11 Ongoing development

Currently, the most highly developed MOR grammar is the one for English, which achieves 99.18% accuracy in tagging for productions from adult native speakers in databases such as CHILDES and AphasiaBank. It is more difficult to reliably determine the accuracy of tagging for child utterances, particularly at the youngest ages when there are often ambiguities in one-word and two-word utterances (Bloom, 1973) that even human coders cannot resolve. MOR grammars are also highly evolved for Spanish, French, German, Mandarin, Japanese, and Cantonese, achieving over 95% accuracy for these languages too. Apart from accuracy of tagging, there is the issue of lexical coverage. For child language, lexical coverage is largely complete for these languages. However, as we deal with more advanced adult productions, such as the academic language in the MICASE corpus or the juridicial language in the SCOTUS corpus, we often need to add new items to the lexicons for given language. Accuracy of grammatical relation tagging by GRASP is highly dependent on accurate tagging by MOR. However, even with accurate MOR tagging, GRASP tagging is at 93% accuracy for English.

3 Running the Program Chain

It is possible to construct a complete automatic morphosyntacgtic analysis of a series of CHAT transcripts through a single command in CLAN, once you have the needed programs in the correct configuration. This command runs the MOR, POST, POSTMORTEM, and MEGRASP commands in an automatic sequence or chain. To do this, you follow these steps:

1. Place all the files you wish to analyze into a single folder.
2. Start the CLAN program (see the Part 2 of the manual for instructions on installing CLAN).
3. In CLAN's **Commands** window, click on the buttom labelled **Working** to set your working directory to the folder that has the files to be analyzed.
4. Under the File menu at the top of the screen, select **Get MOR Grammar** and select the language you want to analyze. To do this, you must be connected to the Internet. If you have already done this once, you do not need to do it again. By default, the MOR grammar you have chosen will download to your desktop.
5. If you choose to move your MOR grammar to another location, you will need to use the **Mor Lib** button in the Commands window to tell CLAN about where to locate it.
6. To analyze all the files in your Working directory folder, enter this command in the Comands window: mor *.cha
7. CLAN will then run these programs in sequence: MOR, POST, POSMORTEM, and MEGRASP. These programs will add %mor and %gra lines to your files.
8. If this command ends with a message saying that some words were not recognized, you will probably want to fix them. If you do not, some of the entries on the %mor line will be incomplete and the relations on the %gra line will be less accurate. If you have doubts about the spellings of certain words, you can look in the 0allwords.cdc file this is included in the /lex folder for each language. The words there are listed in alphabetical order.
9. To correct errors, you can run this command: mor +xb *.cha.. Guidelines for fixing errors are given in chapter 4 below.

4 Morphological Analysis

4.1 The Design of MOR

The computational design of MOR was guided by Roland Hausser's (1990) MORPH system and was implemented by Mitzi Morris. Since 2000, Leonid Spektor has extended MOR in many ways. Christophe Parisse built POST and POSTTRAIN (Parisse & Le Normand, 2000). Kenji Sagae built MEGRASP as a part of his dissertation work for the Language Technologies Institute at Carnegie Mellon University (Sagae, MacWhinney, & Lavie, 2004a, 2004b). Leonid Spektor then integrated the program into CLAN.

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 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 corpus. To maximize the complete analysis of regular formations, only stems are stored in the lexicon and inflected forms appropriate for each stem are compiled at run time.

4.2 Example Analyses

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:   oops I spilled it.
%mor:   co|oops pro:subj|I v|spill-PAST pro:per|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 command, using the MOR grammar for English, which can be downloaded using the **Get MOR Grammar** function described in the previous chapter. The command for running MOR by itself without running the rest of the chain is: mor +d *.cha. After running MOR, the file looks like this:

```
*CHI:  oops I spilled it .
%mor:   co|oops pro:subj|I part|spill-PASTP^v|spill-PAST pro:per|it
.
```

In the %mor tier, words are labeled by their syntactic category or "scat", followed by the pipe separator |, followed then by the stem and affixes. Notice that the word "spilled" is initially ambiguous between the past tense and participle readings. The two ambiguities are separated by the ^ character. To resolve such ambiguities, we run a program called POST. The command is just "post *.cha" After POST has been run, the %mor line will only have v|spill-PAST.

Using this disambiguated form, we can then run the MEGRASP program to create the representation given in the %gra line below:

```
*CHI:   oops I spilled it .
%mor:   co|oops pro:subj|I v|spill-PAST pro:per|it .
```

```
%gra:   1|3|COM 2|3|SUBJ 3|0|ROOT 4|3|OBJ 5|3|PUNCT
```

In the %gra 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. The verb is the Root and it is related to the "left wall" or item 0.

4.3 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 [///].

4.4 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.

+xl Run MOR in the lexicon building mode. This mode takes a series of .cha files as input and outputs a small lexical file with the extension .ulx with entries for all words not recognized by MOR. This helps in the building of lexicons.

+xb check lexicon mode, include word location in data files

+xa check lexicon for ambiguous entries
+xc check lexicon mode, including capitalized words
+xd check lexicon for compound words conflicting with plain words
+xp check lexicon mode, including words with prosodic symbols
+xy analyze words in lex files

4.5 Categories and Components of MOR

MOR breaks up words into their component parts or morphemes. In a relatively analytic language like English, many words require no analysis at all. However, even in English, a word like "coworkers" can be seen to contain four component morphemes, including the prefix "co", the stem, the agential suffix, and the plural. For this form, MOR will produce the analysis: co#n:v|work-AGT-PL. This representation uses the symbols # and – to separate the four different morphemes. Here, the prefix stands at the beginning of the analysis, followed by the stem (n|work), and the two suffixes. In general, stems always have the form of a part of speech category, such as "n" for noun, followed by the vertical bar and then a statement of the stem's lexical form.

To understand the functioning of the MOR grammar for English, the best place to begin is with a tour of the files inside the ENG 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 the stems and affixes in the lexical files.
2. cr.cut – These are the rules that specify the possible combinations of morphemes going from left to right in a 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. post.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 folder contains many files listing the stems and affixes of the language. We will examine it in greater detail below.
8. sf.cut – This file tells MOR how to deal with words that end with certain special form markers such as @b for babbling.

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 glance through these various files to understand their contents.

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. 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. For languages that already have a MOR grammar, 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 on morphosyntactic coding, or if you are trying to write a new set of grammars for some language.

4.6 MOR Part-of-Speech Categories

The final output of MOR on the %mor line uses two sets of categories: part-of-speech (POS) names and grammatical categories. To survey the part-of-speech names for English, we can take a look at the files contained inside the /lex folder. These files 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. This means that it is treated just as and adjective (adj) by the MOR and GRASP programs. In English, the part of speech of a compound is usually the same as that of the last component of the compound. A few additional part of speech (POS) categories are introduced by the 0affix.cut file. These include: n-cl (noun clitic), v-cl (verb clitic), part (participle), and n:gerund (gerund). Additional categories on the %mor line are introduced from the special form marker file called sf.cut. The meanings of these various special form markers are given in the CHAT manual. Also, the punctuation codes end, beg, and cm are the POS codes used for the special character marks given in the punct.cut file.

File (.cut)	POS	ınction	Example
0aae	mixed	African American	finna
0affix	mixed	prefixes and suffixes	see list below
0uk	mixed	terms local to the UK	fave, doofer,
0sing	mixed	Singapore English	bochup, dabao
adj-baby	adj	baby talk adjectives	dipsy, yumsy
adj-irr	adj	irregular adjectives	better, furthest
adj-num	adj	ordinal numerals	eleventh
adj-pred	adj:pred	predicative adjectives	abreast, remiss
adj-sci	adj	scientific terms	biogenous
adj-under	adj	combined adjectives	close_by, lovey_dovey
adj	adj	regular adjectives	tall, redundant
adj+adj+adj	adj	compounds	halfhearted, hot_crossed
adj+adj+adj(on)	adj	compounds	super_duper, easy_peasy
adj+adj+n	adj	compounds	all-time
adj+adj+part	adj	compounds	wholehearted
adj+n+adj	adj	compounds	dogeared, stir_crazy
adj+n+part	adj	compounds	bottlefed
adj+v+*	adj	compounds	various
adv-cp	adv	can be adj or adv	fast, hard
adv-ptl	adv	verbal particles	up, out
adv-tem	adv	temporals	tomorrow, tonight
adv-under	adv	combined adverbs	how_about, as_well
adv-wh	adv:wh	wh term	where, why
adv	adv	regular adverbs	ajar, fast, mostly
adv+adj+n	adv	compounds	heavy_duty
adv+n+adv	adv	compounds	piggyback
adv+n+prep	adv	compounds	thumbs_up
adv+n+prep+n	adv	compounds	face_to_face
adv+prep+n	adv	compounds	offstage
co-biling	co	Cantonese forms	wo, wai, la
co-voc	co	vocatives	honey, dear, sir
co-rhymes	co	rhymes, onomatopoeia	cock_a_doodle_doo
co_under	co	multiword phrases	by_jove, gee_whiz
co	co	regular co	blah, byebye, gah
comp	comp	complementizers	that, which
conj-under	conj	combined conj	even_though, in_case_that
conj	conj	conjunctions	and, although, because
coord	coord	coordinators	and, neither, or
det-art	det:art	deictic determiners	a, the
det-dem	det:dem	demonstratives	this, that
det-int	det:int	interrogatives	what, whose
det-num	det:num	cardinals	two, twelve
det-poss	det:poss	possessives	his, their

n-abbrev	n	abbreviations	c_d, t_v, w_c
n-baby	n	babytalk forms	passie, wawa, booboo
n-dashed	n	noun combinations	cul_de_sac, seven_up
n-dup	n	varying doubles	bibble_wibble, sucky_suck
n-hyphen	n	hyphenated	pre-op, cul-de-sac
n-irr	n	irregular nouns	children, cacti, teeth
n-let	n	all but a and i	c, k, q
n-pluraletant	n:pt	nouns with no singular	golashes, kinesics, scissors
n-sci	n	scientific terms	ampicillin
n-under	n	not compounds	band_aid, beck_and_call
n	n	regular nouns	dog, corner, window
n+adj+n	n	compounds	big+shot, cutie+pie
n+n+conj+n	n	compounds	four+by+four, dot+to+dot
n+n+n-on	n	compounds	quack+duck, moo+cow
n+n+n	n	compounds	candy+bar, foot+race
n+n+novel	n	compounds	children+bed, dog+fish
n+n+part	n	compounds	homemaking
n+n+prep+det+n	n	compounds	corn_on_the_cob
n+on+on-baby	n	compounds	wee+wee, meow+meow
n+other.cut	n	compounts	hide+and+seek
n+v+det+n	n	compounds	follow+the+leader
n+v+n	n	compounds	squirm+worm, snap+bead
n+v+prep	n	compounds	chin_up, hide+out
nonce.cut	various	jabberwocky	mimsy, wabe
n-verb.cdc	n	usually verbs	find, run
on	on	onomatopoeia	boom, choo_choo
on+on+on	on	compounds	cluck+cluck, knock+knock
post	post	post-modifiers	all, too
prep-under	prep	combined prepositions	out_of, in_between
prep	prep	prepositions	under, minus
pro-dem	pro:dem	demonstratives	this, that
pro-exist	pro:exist	existentials	here, there
pro-indef	pro:indef	indefinites	everybody, few
pro-int	pro:int	interrogative	what, who
pro-obj	pro:obj	object case	him, me
pro-per	pro:per	case neutral	it, you
pro-poss	pro:poss	possessives	hers, mine
pro-refl	pro:refl	reflexives	myself, himself
pro-sub	pro:sub	subject case	she, they
punct	end, begin	special marks	‡ and „
quan	qn	quantifier	some, all, only, most
rel	rel	relativizers	that, which
small	inf, neg	small forms	not, to, xxx, yyy
v-aux	aux	auxiliaries	had, getting

v-baby	v	baby verbs	wee, poo
v-clit	v	cliticized forms	gonna, looka
v-cop	cop	copula	be, become
v-hyphen	v	hyphenated forms	x-ray, ad-lib
v-irr	v	irregular verbs	came, beset, slept
v-mod-aux	mod	modal auxiliaries	hafta, gotta
v-mod-inf	mod, v, part	cliticized with inf	hafta, gonna
v-mod	mod	modals	can, ought
v-sci	v	scientific forms	protonate
v	v	regular verbs	run, take, remember
v+adj+v	v	compounds	deep-fry, tippytoe
v+n+v	v	compounds	bunny+hop, sleepwalk
v-noun	v	usually nouns	father, shoe
zero	0x	omitted words	0adj, 0n, 0subj

The construction of these lexicon files involves a variety of decisions. Here are some of the most important issues to consider.

1. Words may often appear in several files. For example, virtually every noun in English can also function as a verb. However, when this function is indicated by a suffix, as in "milking" the noun can be recognized as a verb through a process of morphological derivation contained in a rule in the cr.cut file. In such cases, it is not necessary to list the word as a verb. Of course, this process fails for unmarked verbs. However, it is generally not a good idea to represent all nouns as verbs, since this tends to overgenerate ambiguity. Instead, it is possible to use the POSTMORTEM program to detect cases where nouns are functioning as bare verbs.
2. If a word can be analyzed morphologically, it should not be given a full listing. For example, since "coworker" can be analyzed by MOR into three morphemes as co#n:v|work-AGT, it should not be separately listed in the n.cut file. If it is, then POST will not be able to distinguish co#n:v|work-AGT from n|coworker.
3. The zero-cut file lists possible omitted parts of speech, such as 0det. However, in the transcript, this could be represented as either 0det or 0the.
4. It is always best to use spaces to break up word sequences that are 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 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. The pro:int category is used to cover both initial wh-words in utterances such as *why are you smiling*, as well as headless clauses such as *I know why you are*

smiling.

4.7 MOR Grammatical Categories

In addition to the various part-of-speech categories provided by the lexicon, MOR also inserts a series of grammatical categories, based on the information about affixes in the 0affix.cut file, as well as information inserted by the a-rules and c-rules. If the category is regularly attached, it is preceded by a dash. If it is irregular, it uses an amerpsand. For English, the inflectional categories are:

Abbreviation	Meaning	Example	Analysis
PL	nominal plural	cats	n|cat-PL
PAST	past tense	pulled	v|pull-PAST
PRESP	present participle	pulling	v|pull-PRESP
PASTP	past participle	broken	v|break-PASTP
PRES	present	am	cop|be&1S&PRES
1S	first singular	am	cop|be&1S&PRES
3S	third singular present	is	cop|be&3S&PRES
13S	first and third	was	cop|be&PAST&13S

In addition to these inflectional categories, English uses these derivational morphemes:

Abbreviation	Meaning	Example	Analysis
CP	comparative	stronger	adj|strong-CP
SP	superlative	strongest	adj|strong-SP
AGT	agent	runner	n|run&dv-AGT
DIM	diminutive	doggie	n|dog-DIM
FUL	denominal	hopeful	adj|hope&dn-FULL
NESS	deadjectival	goodness	n|good&dadj-NESS
ISH	denominal	childish	adj|child&dn-ISH
ABLE	deverbal	likeable	adj|like&dv-ABLE
LY	deadjectival	happily	adj|happy&dadj-LY
Y	deverbal, denominal	sticky	adj|stick&dn-Y

In these examples, the features dn, dv, and dadj indicate derivation of the forms from nouns, verbs, or adjectives.

Other languages use many of these same features, but with many additional ones, particularly for highly inflecting languages. Sometimes these are lowercase and sometimes upper. Here are some examples:

Affix	Meaning	Affix	Meaning	Affix	Meaning
KONJ	subjunctive	ADV	adverbial	m	masculine
SUB	subjunctive	SG	singular	f	feminine
COND	conditional	PL	plural	AUG	augmentative
NOM	nominative	IMP	imperative	PROG	progressive

ACC	accusative	IMPF	imperfective	PRET	preterite
DAT	dative	FUT	future		
GEN	genitive	PASS	passive		

4.8 Compounds and Complex Forms

The lexical files include many special compound files such as n+n+n.cut or v+n+v.cut. 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 many forms that resemble compounds, but which are best viewed as units with non-morphemic components. For example, kool_aid and band_aid are not analytic combinations of morphemes, although they clearly have two components. The same is true for hi_fi and coca_cola. The underscore character 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. If 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.
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". It is worth noting here that German treats all such forms as single words. This means that different conventions must 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. They should not be treated as 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 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. 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 overall part of speech of the compound, 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.

4.9 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 for 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 standard pronoun. However, since the second word 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).

4.10 Affixes

The inflectional and derivational affixes of English are listed 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.

4.11 Control Features and Output Features

The lexical files include several control features that specify how stems should be treated. One important set includes the [comp x+x] features for compounds. This feature controls 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. Features like [block ed] are used to prevent reocognition of overregularized forms such as *goed*.

There are also a variety of features that are included in lexical entries, but not necessarily present in the final output. For example, the feature of gender is used to determine patterns of suffixation in Spanish, but to include this feature in the output it must be present and not commented in the output.cut file. Other lexical features of this type include root, ptn, num, tense, and deriv.

5 Correcting errors

When running MOR on a new 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 see if all the words follow basic CHAT rules, such as not including numbers or capital letters in the middle of words. There are several common reasons for a word not being recognized:

1. It is misspelled. If you have doubts about the spellings of certain words, you can look in the 0allwords.cdc file this is included in the /lex folder for each language. The words there are listed in alphabetical order.
2. The word should be preceded by and ampersand & to block look up through MOR. There are four forms using the ampersand. Nonwords just take the & alone, as in &gaga. Incomplete words should be transcribed as &+text, as in &+sn for the beginning of *snake*. Filler words should be transcribed as &-uh. Finally, sounds like laughing can be transcribed as &=laughs, as described more extensively in the CHAT manual.
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 stem is in the lexicon, 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 macw@cmu.edu.
7. 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 macw@cmu.edu 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 command with +xb or +xl and get a list of unrecognized words.

To correct these problems, 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 know that a corpus passes CHECK, you will want to see whether it contains words that are either misspelled or not yet in the MOR lexicon. You do this by running the command:

```
mor +xb *.cha
```

The output from this command will have the extension .ulx.cex. After running the command, its name will appear at the end of the output in the **CLAN Output** window. If that window tells you that "all words were found in the lexicon", then you can proceed with running

```
mor *.cha
```

However, if not all words are recognized, you can triple-click on the line listing ther "Output File" and it will open the list of words not yet recognized by MOR. In any large corpus, is extremely unlikely that every word 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 irregular forms of a word as well as the stems of regular 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 a series of files each with a series of lexical entries with 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.

5.1 Lexicon Building

When running the mor +xb command, you may wish to run the command in the form mor +xl. The +xb form lists each separate token of an unrecognized word, whereas the +xl form combines all the tokens into a single type. The advantage of the +xb format is that you can click on each occurrence and change it. However, for very common errors, the +xl format is useful because it will allow you to see what forms should be changed globally using the CHSTRING command.

When working with the output of ther +xb form, you must then go through this file and determine whether to discard, complete, or modify each missing case. For example, it may be impossible to decide what category "ta" belongs to without examining where it

occurs in the corpus. In this example, a scan of the Sarah files in the Brown corpus (from which these examples were taken), reveals that "ta" is a variant of the infinitive marker "to":

```
*MEL:   yeah # (be)cause if it's gon (t)a be a p@l it's
         got ta go that way.
```

This missing form can be repaired by joining got and ta into gotta, because that form is listed in the lexicon. Alternatively, the sequence can be coded as here:

```
*MEL:   yeah # (be)cause if it's gon (t)a be a p@l it's
         gotta [: got to] go that way.
```

Another common source of error is misspelling, which can be corrected.

In many other cases, you will find that some words are just missing from the lexicon. For these, you can create a file with a name like 0morewords.cut which you add to the files in /lex. After doing this, please send the contents of this file to macw@cmu.edu, so that I can add these missing words to the authoritative version of the lexicon.

5.2 Disambiguator Mode

When POST works smoothly, there is littlel need for hand disambiguation. However, ambiguities within a given part of speech cannot be resolved by POST and must be disambiguated by hand using Disambiguator Mode. Also, when developing POST for a new language, you may find this tool useful. Toggling the **Disambiguator Mode** option in the **Mode** menu (escape-2) 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."

When opening a new file for disambiguation using the escape-2 function, make sure that your cursor is at the beginning of the file. If your cursor is placed after the last ambiguity, the function will say that you are "Done" even though ambiguities still remain.

By default, Disambiguator Mode is set to work on the %mor tier. However, you may find it useful for other tiers as well. To change its tier setting, select the **Edit** menu and pull down to **Options** to get the **Options** dialog box. Set the disambiguation tier to the tier you want to disambiguate. To test all of this out, edit the sample.cha file, reset your default tier, and then type *Esc-2*. The editor should take you to the second %spa line which has:

```
%spa:   $RES:sel:ve^$DES:tes:ve
```

At the bottom of the screen, you will have a choice of two options to select. Once the correct one is highlighted, you hit a carriage return and the correct alternative will be inserted. If you find it impossible to decide between alternative tags, you can select the UND or undecided tag, which will produce a form such as "und|drink" for the word drink, when you are not sure whether it is a noun or a verb.

6 A Formal Description of the Rule Files

Users working with languages for which grammar files have already been built do not need to concern themselves with this section or the next. However, users who need to develop grammars for new languages or who find they need 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 from the English MOR grammar.

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.

6.1 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. For a clause 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 to give a more general description of the shape of a string.

6.2 Pattern-matching symbols

The specification of orthographic patterns relies on a set of symbols derived from the regular expression (regexp) system in Unix. The rules of this system are:

1. The metacharacters are: * [ ] | . ! All other characters are interpreted literally.
2. A pattern that contains no metacharacters will only match itself, for example the

pattern “abc” will match only the string “abc”.

3. The period matches any character.
4. The asterisk * allows any number of matches (including 0) on the preceding character. For example, the pattern '.*' will match a string consisting of any number of characters.
5. The brackets [ ] are used to indicate choice from among a set of characters. The pattern [ab] will match either a or b.
6. A pattern may consist of a disjunctive choice between two patterns, by use of the | symbol. For example, the pattern will match all strings which end in x, s, sh, or ch.
7. It is possible to check that some input does not match a pattern by prefacing the entire pattern with the negation operator !.

6.3 Variable notation

A variable is used to name a regular expression and to record patterns that match it. A variable must first be declared in a special variable declaration statement. Variable declaration statements have the format: “VARNAME = regular-expression” where VARNAME is at most eight characters long. If the variable name is more than one character, this name should be enclosed in parenthesis when the variable is invoked. Variables are particularly important for the arules in the ar.cut file. In these rules, the negation operator is the up arrow ^, not the exclamation mark. Variables may be declared through combinations of two types of disjunction markers, as in this example for the definition of a consonant cluster in the English ar.cut file:

```
O = [^aeiou]|[^aeiou][^aeiou]|[^aeiou][^aeiou][^aeiou]|qu|sq
```

Here, the square brackets contain the definition of a consonant as not a vowel and the bar or turnstile symbols separate alternative sequences of one, two, or three consonants. Then, for good measure, the patterns “qu” and “squ” are also listed as consonantal onsets. For languages that use combining diacritics and other complex symbols, it is best to use the turnstile notation, since the square bracket notation assumes single characters. In these strings, it is important not to include any spaces or tabs, since the presence of a space will signal the end of the variable.

Once declared, the variable can be invoked in a rule by using the operator $. If the variable name is longer than a single character, the variable name should be enclosed in parentheses when invoked. For example, the statement `X = .*` declares and initializes a variable named “X.” The name X is entered in a special variable table, along with the regular expression it stands for. Note that variables may not contain other variables.

The variable table also keeps track of the most recent string that matched a named pattern. For example, if the variable X is declared as above, then the pattern $Xle will match all strings that end in *le.* For example, the string *able* will match this pattern, because *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.

6.4 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.

6.5 Arules

The function of the arules in the arules.cut file and the additional files in the /ar folder 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 variable statement, 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.
3. ALLOSTEM is used to produce an output stem. This action is not necessary in this example, because this rule is fully regular and produces a noninflected stem. However, the arule that converts "postman" into "postmen" uses this ALLOSTEM action:

   ALLOSTEM = $Xman&PL

   The result of this action is the form postman&PL that is placed into the %mor line without the involvement of any of the concatenation rules.

There are two special category 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 m] for the Spanish word "hombre" will end up producing n|hombre&m. The entry [proc dim] for Chinese reduplicative verbs wil end up producing v|kan4-DIM for the reduplicated form kan4kan4. These methods allow allorules to directly influence the output of MOR.

Every set of action statements leads to the generation of an additional allomorph for the online lexicon. Thus, if an arule clause contains several sets of action statements, each labeled by the header ALLO:, then that arule, when applied to one entry from the disk lexicon, will result in several entries in the online lexicon. To create the online lexicon, the arules are applied to the entries in the disk lexicon. Each entry is matched against the arules in the order in which they occur in the arules file. This ordering of arules is an extremely important feature. It means that you need to order specific cases before general cases to avoid having the general case preempt the specific case.

As soon as the input matches all conditions in the condition section of a clause, the actions are applied to that input to generate one or more allos, which are loaded into the on-line lexicon. No further rules are applied to that input, and the next entry from the disk lexicon is then read in to be processed. The complete set of arules should always end with a default rule to copy over all remaining lexical entries that have not yet been matched by some rule. This default rule must have this shape:

```
% default rule- copy input to output
RULENAME: default
LEX-ENTRY:
ALLO:
```

6.6 Crules

The purpose of the crules in the crules.cut file 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:

```
keyword function
STARTSURF        check surface of start input against some pattern
STARTCAT         check start category information
NEXTSURF         check surface of next input against some pattern
NEXTCAT check next category information
MATCHCAT         check that start and next match for a feature-
value pair type
RESULTCAT        output category information
```

Here is an example of a piece of a rule that uses most of these keywords:

```
S = .*[sc]h|.*[zxs] % strings that end in affricates
O = .*[^aeiou]o % things that end in o
% clause 1 - special case for "es" suffix
 if
 STARTSURF = $S
 NEXTSURF = es|-es
 NEXTCAT = [scat vsfx]
 MATCHCAT [allo]
 then
 RESULTCAT = STARTCAT, NEXTCAT [tense], DEL [allo]
 RULEPACKAGE = ()
```

This rule is used to analyze verbs that end in -es. There are four conditions that must be matched in this rule:

1. The STARTSURF is a stem that is specified in the declaration to end in an affricate. The STARTCAT is not defined.
2. The NEXTSURF is the -es suffix that is attached to that stem.
3. The NEXTCAT is the category of the suffix, which is "vsfx" or verbal suffix.
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 CYTPE marker is used to indicate which concatenation symbol is used when concatenating the morphemes together into a parse for a word. The # 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 insert a suffix after the stem.

Rules with CTYPE START are 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 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.

7 Building new MOR grammars

7.1 minMOR

The simplest possible form of a MOR grammar is represented in the "min" grammar that you can download from the MOR grammars page at https://childes.talkbank.org.

You can begin your work using with the sample minimal MOR grammars available from the net. This grammar includes

1. the sf.cut file that all of the MOR grammars use,
2. a sample.cha file with a few words
3. a basically blank ar.cut file, because no allomorphy is yet involved,
4. a cr.cut file that recognizes the parts of speech you will create, along with one rule for making plural nouns, and
5. a lex folder with examples of verbs, nouns, a determiner, and a suffix.

You can adjust this format to your language and use a different sample.cha file to test out the operation of this minimal MOR grammar. This system 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.

You can test out your grammar either by running mor +d sample.cha or else using interactive MOR with mor +xi. For example, if you use interactive MOR and type "dogs" you should get

```
Result: n|dog-PL
```

7.2 Adding affixes

At some point, you will realize that it would be more efficient to create a system for lexical analysis, rather than relying only on full forms. This will require you to build up a morphological grammar. When building a morphology for a new language, it is best to begin with a paper-and-pencil analysis of the 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. For example, we have used the Berlitz conjugation books for French, German, Italian, and Spanish.

Once this basic groundwork is finished, 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.

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.

7.3 Interactive MOR

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.

7.4 Testing

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. These files provide what computer scientists call "regression testing". 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 and +xb. 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, and having a fast machine will speed this process.

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 may need to 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 must 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.

7.5 Building Arules

In English, the main arule 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 OV$C characterizes verbs like “bat” that end with vowels followed by consonants. The first allo will produce words like “batting” or “batter” and the second will give a stem for “bats” or “bat”. A complete list of allomorphy types for English is given in the file engcats.cdc in the /docs folder in the MOR grammar.

When a user types the “mor” command to CLAN, the program loads up all the *.cut files in the lexicon and then passes each lexical form past the rules of the ar.cut file. The rules in the ar.cut file are strictly ordered. If a form matches a rule, that rule fires and the allomorphs it produces are encoded into a lexical tree based on a “trie” structure. Then MOR moves on to the next lexical form, without considering any additional rules. This means that it is important to place more specific cases before more general cases in a standard bleeding relation. There is no “feeding” relation in the ar.cut file, since each form is shipped over to the tree structure after matching.

7.6 Building crules

The other "core" file in a MOR grammar is the cr.cut file that contains the rules that specify pathways through possible words. The basic idea of crules or concatenation or continuation rules is taken from Hausser's (1999) left-associative grammar which specifies the shape of possible "continuations" as a parser moves from left to right through a word. Unlike the rules of the ar.cut file, the rules in the cr.cut file are not ordered. Instead, they work through a "feeding" relation. MOR goes through a candidate word from left to right to match up the current sequence with forms in the lexical trie 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#part|consider-PRESP^re#n:gerund|consider-GERUND
```

Later, POST will work to choose between these two possible readings of "reconsidering" on the basis of the syntactic context. As we noted earlier, when "reconsidering" follows an auxiliary ("is eating") or when it functions adjectivally ("an eating binge"), it is treated as a participle. However, when it appears as the head of an NP ("eating is good for you"), it is treated as a gerund. Categories and processes of this type can be modified to match up with the requirements of the GRASP program to be discussed below.

The process of building ar.cut and cr.cut files for a new language involves a slow iteration of lexicon building with rule building. 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 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.

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.

8 MOR for Bilingual Corpora

It is possible to use MOR and POST to process bilingual corpora automatically. 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. 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. For the YipMatthews English files, the steps are:

1. To make sure all words in English are recognized, set the MOR library to ENG and run: mor -s"[- yue]" +xb *.cha
2. Fix any errors and add any missing words to the ENG lexicon.
3. Run: mor -s"[- yue]" *.cha
4. Run CHECK to check for problems.
5. To make sure all words in Cantonese are recognized, set the MOR library to YUE

and run: mor +s"[- yue]" +xb *.cha
6. Fix any errors and add any missing words to the YUE lexicon.
7. Run: mor +s"[- yue]" *.cha
8. 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:wh|where v|be n|grandma adv|first v|tell pro|me ?
*LIN:   well, what's this ?
%mor:   co|well pro:wh|what~v|be pro:dem|this ?
*CHI:   [- yue] xxx 呢 個 唔 夠 架 .
%mor:   unk|xxx det|ni1=this cl|go3=cl neg|m4=not adv|gau3=enough
        sfp|gaa3=sfp .
*LIN:   [- yue] 呢 個 唔 夠 .
%mor:   det|ni1=this cl|go3=cl neg|m4=not adv|gau3=enough .
*LIN:   <what does it mean> [>] ?
%mor:   pro:wh|what v:aux|do pro|it v|mean ?
```

This type of analysis is possible whenever MOR grammars exist for both languages, as would be the case, for example, for Japanese-English, Spanish-English, English-German, German-French, Spanish-French, Mandarin-Cantonese, or Italian-Mandarin bilinguals.

9 POST

POST was written by Christophe Parisse of INSERM, Paris for 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. It uses a database called post.db 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.

There are POST databases now for Cantonese, Danish, English, French, German, Hebrew, Italian, Japanese, Mandarin, and Spanish. As our work with POST progresses, we will make these available for additional languages. If you wish to create a post.db yourself for a new language, please look at the instructions for POSTTRAIN in a later section of this chapter. To run POST, you can use this command format :

```
post *.cha
```

However, POST will also run automatically after running MOR, unless you add the +d option to MOR. During this process, MOR runs first, then the rules in the prepost.cut file operate, then POST runs, and finally the rules in postmortem.cut apply.

The accuracy of disambiguation by POST for English, along with prepost.cut and postmortem.cut, is between 95 and 97 percent. This means that there will be some errors.

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)
+lN when followed by a number the +l switch controls the number of output lines

+unk tries to process unknown words.

+sN N=0 (default) replace ambiguous %mor lines with disambiguated ones
N=1 keep ambiguous %mor lines and add disambiguated %pos lines.
N=2 output as in N=1, but with slashes marking undecidable cases.
N=3 keep ambiguous %mor lines and add %pos lines with debugging info.
N=4 inserts a %nob line before the %mor/%pos line that presents the results of the analysis without using Brill rules.
N=5 outputs results for debuging POST grammars.
N=6 complete outputs results for 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
+/-t#Target_Child - select target child's tiers
+/-t@id="*|Mother|*" - select mother's tiers

9.1 POSTLIST

POSTLIST provides a list of tags used by POST. It is run on the post.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.

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-PRESP^v:n|play-PRESP => 3 [0,6,0] means that play has 3 potential categories n:gerund|play-GERUND and part|play-PRESP and v:n|play-PRESP and that it was found 6 time in the second category in the training corpus.

9.2 POSTMODRULES

This program outputs the rules used by POST for debugging rules. POSTMODRULES is used to check and modify Brill rules after initial training.

9.3 PREPOST

PREPOST is not a separate program, but rather a filter that applies during the operation of MOR. It relies on a file called prepost.cut to automatically prune down the number of alternative readings created by MOR even before the POST program operates. These rules assume that the relevant syntactic environment can fully determine the selection without attention to further information. To illustrate how the rules operate, consider the case of the English final *'s* suffix which could be a possessive, a copula, or an auxiliary. So, the word *Ben's* have these ambiguities:

```
n:prop|Ben~aux|be&3S^n:prop|Ben~cop|be&3S^adj|Ben&dn-POSS
```

However, when this word precedes a noun, it is always a possessive. So, the following PREPOST rule will select this reading before POST runs:

```
adj|*-POSS^* n|* => adj|*-POSS n|*
```

In this rule, the term adj|*-POSS^* indicates any form that has adj|*-POSS as one reading, because the ^* part says that any other readings are possible. It is also possible to compose more complex rules that include the *^* terms to indicate any number of intervening items, as in this example for processing *to clean the floor and wash the dishes*

```
v|*^* *^* coord|and v|*^* => v|* *^* coord|and v|*
```

Here are some additional examples:

```
# "adj|*" and any number of other ambiguity choices "^*"
det:art|* adj|*^* n|* => det:art|* adj|* n|*

# any number of any "*^*" elements in between
det:dem|*^* *^* n|*^* => det:dem|* n|*

# any number of "adv|*^*" elements in between
aux|*^cop|*  (adv|*^*)  part|*-PRESP^*  =>  aux|*  (adv|*^*)  part|*-
PRESP

# next to first element or last element
$b $e

# "$-" means do no change that one element on replacement side at
all
pro:sub|*^* prep|*^adv|* prep|*^adv|* => pro:sub|* $- $-

# skip exactly one element
aux|*^cop|* * part|*-PRESP^* => aux|* $- part|*-PRESP

# skip exactly two elements
aux|*^cop|* * * part|*-PRESP^* => aux|* $- $- part|*-PRESP

# $- with no space inserts a form that wasn't in the input
prep|*^!conj|*^* adv|here^* => prep|* $-n|here
```

The rules of both PREPOST and POSTMORTEM use the following syntax:

```
#************************************
# Instructions:
# '~'     in the rule means clitic in the utterance is not treated as space, otherwise it is
# !       used before element refers to the whole element
# ,!      used within element after "," refers only to that choice
# $-      means copy whatever search pattern matched in full and unchanged to the output
# (...)   means match choices within zero or any number of times; the "to" part must be ()
# [...]   means match choices within zero or only one time; the "to" part must be []
# pro:*|*,n|*,!pro:dem|*        means match "pro:*|" or "n|" once, but fail at "pro:dem|";
#                               the "to" part must be $-
# !part|*P*P    means match only if part|*P*P not found; the "to" part must be !-
# \       means the rule continues on the next line
#************************************
```

9.4 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.

The rules in POSTMORTEM vary markedly from language to language. They are particuarly important for German, where they are used to flesh out the morphological features of the noun phrase.

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 to make each replacement on a case by case basis.

9.5 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 post.db. 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 +d on your training files. This will produce an ambiguous %mor line. Adding the +d switch is necessary to avoid automatic disambiguation.
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 without the +d switch to create a new disambiguated %mor line to go along with the %trn line.
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 you need to repeatedly cycle through a standard set of commands while making continual changes to the input data. Here is a sample sequence that uses the defaults in POST and POSTTRAIN:

```
mor *.cha +1
posttrain +c +o0err.cut +x *.cha
post *.cha +1
trnfix *.cha
```

In these commands, the +1 must be used carefully, since it replaces the original. If a program crashes or exits while running with +1, the original can be destroyed, so make a backup of the whole directory first before running +1. TRNFIX can be used to spot mismatches between the %trn and %mor lines.

The options for POSTTRAIN are:

+a train word frequencies even on utterances longer than length 3.
+**b** extended learning using Brill's rules
-b Brill's rules training only (However, we never use Brill rules)
+boF append output of Brill rule training to file F (default: send it to screen)
+bN parameter for Brill rules
1- means normal Brill rules are produced (default)
2- means only lexical rules are produced
3- same as +b1, but eliminates rules redundant with binary rules
4- same as +b2, but eliminates rules redundant with binary rules
+btN threshold for Brill rules (default=2). For example, if the value is 2, a rule should correct 3 errors to be considered useful. To generate all possible rules, use a threshold of 0.
+c create new POST database file with the name post.db (default)
+cF create new POST database file with the name F
-c 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 isused 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
+/-t#Target_Child - select target child's tiers
+/-t@id="*|Mother|*" - select mother's tiers

+x use syntactic category suffixes to deal with stem compounds

When using the default switch form of the error log, lines that begin with @ indicate that the %trn and %mor had different numbers of elements. Lines that do not begin with @ represent simple disagreement between the %trn and the %mor line in some category assignment. For example, if %mor has *pro:dem^pro:exist* and %trn has *co* three times. Then +s0 would yield: 3 there co (3 {1} pro:dem (2} pro:exist).

By default, POSTTRAIN uses all the affixes in the language and none of the stems. If you wish to change this behavior, you need to create a file with your grammatical names for prefixes and suffixes or stem tags. 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.

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 our work in constructing the training corpus for POSTTRAIN, we bumped into areas of English grammar where the distinction between parts of speech is difficult to make without careful specification of detailed criteria. Work with other languages will probably run into problems of this general type. We can identify four 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 t*here you are* where *there* could be interpreted as either specifying a location vs. *there is a car* in which *there* is *pro:exist*.

9.6 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).

9.7 TRNFIX

By default, this program compares the forms on the %trn line with those on the %mor line and reports each mismatch. This simple command will run across all the files in a folder of training files:

```
trnfix *.cha
```

You can then triple click on each line for a mismatch to determine whether the error is in the %mor or the %trn line. If it is in the %trn line, you can fix it and that will improve your training and the accuracy of POST. TRNFIX can also be used to compare the %gra and %grt lines using this form:

```
trnfix +b%gra +b%grt *.cha
```

10 GRASP – Syntactic Dependency Analysis

This chapter, with contributions from Eric Davis, Shuly Wintner, Brian MacWhinney, Alon Lavie, Andrew Yankes, and Kenji Sagae, describes a system for coding syntactic dependencies in the English TalkBank corpora. This chapter explains the annotation system and describes each of the grammatical relations (GRs) available for tagging dependency relations.

10.1 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. These dependency relations are unidirectional and cannot be used to represent bidirectional relations. 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 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.

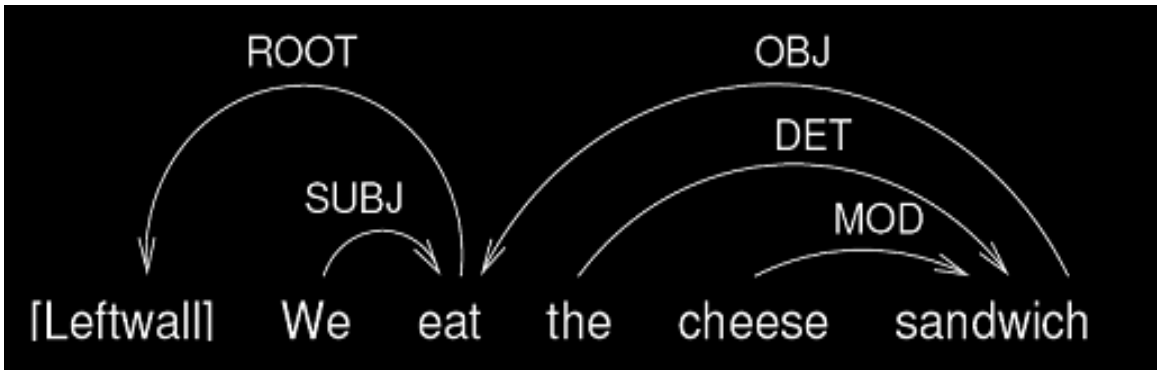
Here is an example of the coding of the sentence *the big dog chased five cats* for dependencies:

```
*TXT:   the big dog chased five cats.
%mor:   det|the adj|big n|dog v|chase-PAST quant|five n|cat-PL.
%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:



This picture is equivalent to this notation in 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

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 large set of grammatical relations (GRs), given below.
3. GRs apply to phrases or clusters of words connected through dependency relations. The most import phrasal types are: noun phrase, verb phrase, absolute phrase, gerund phrase, auxiliary phrase, and infinitival phrase. To achieve attachment to heads, phrases must be represented by one of their component lexical items. In some structures this can be a relativizer or conjunction; in others, it can be the main verb of the subordinate clause.

The following is a comprehensive list of the grammatical relations in the GRASP annotation scheme. Example GRs as well as relations to similar GRs 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.

10.2 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 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, 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. COMPlement identifies a 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. Examples: I think that *was* Fraser. You're going to *stand* on my toe. I told you to *go*. Eve, you stop *throwing* the blocks.
6. PREDicate identifies a predicate nominal or predicate adjective of verbs such as *be* and *become*. The predicate can also be an embedded clause, and then head of that clause is the verb. PRED should not be confused with COMP, which identifies the complement of a verb. If there is a relativizer, it attaches to the verb of the embedded clause through the LINK relation. Examples: I'm not *sure*. He is a *doctor*. This is how I *drink* my coffee. My goal is to *win* the competition.
7. ClausalPrepositionalOBJect = CPOBJ indentifies a full clause that serves as the object of a preposition. The verb of the clause attaches to the preposition which is the head. Here, again, the relativizer attaches to the verb of the subordinate clause through the LINK relation. Ex: I'm confused *about what you are asking*.
8. ClausalOBJect = COBJ indentifies a full clause that serves as OBJ. The verb of the object clause attaches to the main verb which is the head. Here, again, the relativizer attaches to the verb of the subordinate clause through the LINK relation. Ex: I remember *what you said*.
9. PrepositionalOBJect = POBJ is the relation between a preposition and its object. The head is a preposition, and the dependent is typically a noun. The traditional treatment of the prepositional phrase views the object of the preposition as the head of the prepositional phrase. However, we are here treating the preposition as the head, since the prepositional phrase then participates in a further JCT relation to a head verb or a NJCT relation to a head noun. Ex: You want to sit on the *stool*?
10. 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.

10.3 Argument-head relations

Relations in which the arguments (rather than the predicates) serve as the heads include relations of adjunction and modification.

1. 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 these, the adjunct is the predicate, since it opens 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 *way* in *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.
2. ClausalconJunCT = CJCT identifies a finite clause that adjoins coordinatively to a verb, adjective, or adverb head. The dependent is the main verb of the 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. When two clauses are combined with *and* CJCT is the link between the main verb of the second clause and the first. However, this relation should not be used if the subject is not repeated, as in *John runs 5 miles, and swims one every day*. Instead, this is a case of a coordinated verb.
3. 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.
4. 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.
5. 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.
6. POSTMODifier identifies a postposed nominal modifier. The head is a noun, and the dependent is a following adjective. Ex: I will dig a hole *bigger* than a foot. Often these express resultative relations, as in I painted the barn *red*. Sometimes they occur after intervening material, as in He pulled her out of the water *safe*.

7. 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 to avoid problems in processing the verb clitics that take the same phonological shape. Ex: John*'s* book.
8. APPositive identifies the relation between an appositive phrase and its head. Ex: Barack Obama, President of the United States. It can also be used to link a topicalized noun to a following pronoun. Ex: John, he really likes waffles.
9. 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*. This relation can also be used when the head is an adjective. Ex: He was happy *he found the girl*.
10. 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.
11. 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.
12. 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.
13. PostQuantifier = PQ is the relation between a postquantifier and the preceding head nominal. Ex: We *both* arrived late.
14. 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?
15. 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.
16. 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.
17. 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, COBJ, CPOBJ, PRED, or COMP relation. Ex: Wait *until* the noodles are cool.
18. 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.

10.4 Extra-clausal elements

In these relations, the dependent is a clausal modifier. These should be linked to the root.

1. COMmunicator identifies a communicator (such as *hey*, *okay*, etc) or a vocative when it occurs inside an utterance. When these occur at the beginning or end of utterances, then use BEG and END instead. The head of COM is the ROOT.
2. BEGin identifies an initial clause-external element, such as a vocative or topic. 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|begp. The BEGP relation is linked to BEG.
3. END identifies a final clause-external postposed element, including sentence final particles, final vocatives, final interactionals, and single word tags like *right?* Like COM, but unlike BEG, the head of the END is the ROOT. This is done, because the parser works better in this way. END is marked in the main line with a preceding „ mark that is coded on the %mor line as end|endp. Ex: Some more cookies, *Eve*?
4. INCompleteROOT 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*.
5. OMission is used when ellipsis or omission leaves determiners or other modifiers without heads. They should then be attached to the most local predicate using the OM relation.

10.5 Cosmetic relations

There are several relations that are just used during transcription to assist in the accurate training of the GRASP tagger:

1. PUNCTuation is the relation between the final punctuation mark and ROOT.
2. 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".
3. BEGP is the relation between the ‡ mark and the BEG.
4. ENDP is the relation between the „ mark and the END.
5. 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 which 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 some of these, the final element is the head and the initial elements all depend on the final head. However, in coordinations, we take the first element as the head.

1. NAME identifies a string of proper names such as Eric Davis and New York Central Library. The initial name is the dependent, and the following name is the head. Ex: My name is *Tom* Jones.
2. DATE identifies a date with month and year, month and day, or month, day, and year. Examples include October 7, 1980 and July 26. For consistency with compounds and NAME, we regard the final element in these various forms as the head. Ex: *October seventh nineteen* ninety.
3. ENUMeration involves a relation between elements in a series without any coordination based on a conjunction (*and, but, or*). The series can contain letters, numbers, and nominals. The head is the first item in the series, and all the other items in the enumeration depend on this first word. Ex: one, *two*, *three*, *four*.
4. CONJ involves a relation between a coordinating conjunction and the cluster of preceding conjoined items 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.
5. 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. In the *neither X nor Y* and *either X or Y* constructions, the words *neither, nor, either,* and *or* are treated as coord by MOR and each coord words links to the following content word (noun, verb, or adjective) with COORD. Then the first content word links to the second with ENUM.

10.6 MEGRASP

MEGRASP is the CLAN command that creates a %gra line based on the forms in the %mor line. MEGRASP uses a maximum entropy classifier system encoded in the megrasp.mod database file to create a dependency parsing based on binary grammatical relations. To visualize these dependencies graphically, you can triple click on a %gra line and CLAN will run a web service that produces labelled graph structure on your computer screen. The figures in the next chapter are screengrabs taken from those displays.

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.

11 Building a training corpus

Once a megrasp.mod file has been created by running MEGRASP in training mode, the creation of the %gra line follows automatically in the chain of MOR-PREPOST-POST-POSTMORTEM-MEGRASP. However, to produce accurate tagging it is important to have properly tagged GRs in the training corpus. To do this accurately involves making consistent decisions for certain GRs that are easy to confuse. This chapter provides illustrations that can help in making these choices between competing GR assignments. These illustrations were produced using the GraphViz system, which we run as a web service. If your computer is connected to the web, you can double click on a %gra line in a transcript and the service will produce one of these graphs which you can use to verify the accuracy of the grammatical relations.

11.1 ROOT, SUBJ, DET, PUNCT

ROOT identifies the topmost verb of a sentence, which doesn't depend upon any other element in the sentence.

The head of ROOT is 0, the LeftWall.

SUBJect identifies the subject of a verb, as long as that subject is a noun phrase. (If the subject is instead a verbal clause, it's tagged differently – see section 13.)

The head of SUBJ is the verb for which it is acting as a subject.

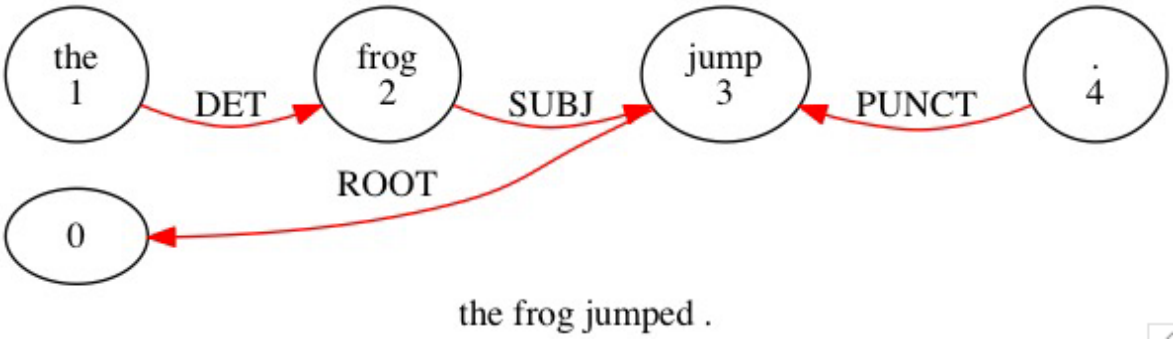
DETeterminer identifies the determiner of a noun phrase. This can be a definite or indefinite article (*the, an*); a possessive pronoun (*my, your*); or a demonstrative (*this, those*). However, it doesn't include quantifiers like *many* and related categories (see section 12).

The head of DET is the noun to which it attaches.

PUNCTuation identifies the punctuation mark at the end of a sentence (for initial and medial punctuation, see section 16).

The head of PUNCT is the ROOT.

Example 1a

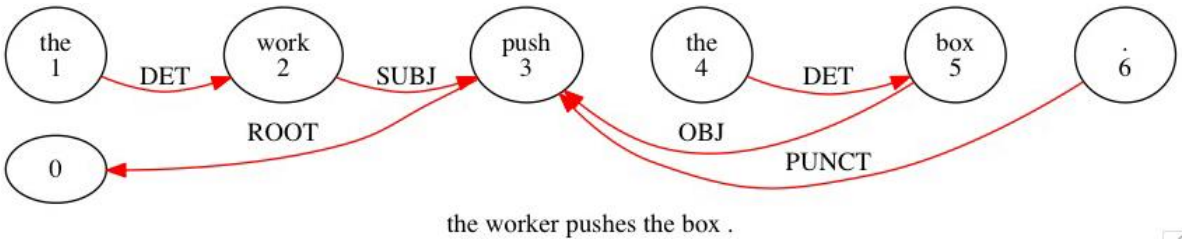


*PAR: the frog jumped .
%mor: det:art|the n|frog v|jump-PAST .
%gra: 1|2|DET 2|3|SUBJ 3|0|ROOT 4|3|PUNCT

Here there is only one verb, *jumped*, clearly functioning as the topmost verb of the sentence. So it's identified as the ROOT and tied to 0, and the PUNCT ties to *jumped* in turn. *frog* serves as the subject of the verb, so it ties to *jumped* as a SUBJ. Finally, *the* ties as a DET to the noun which it specifies.

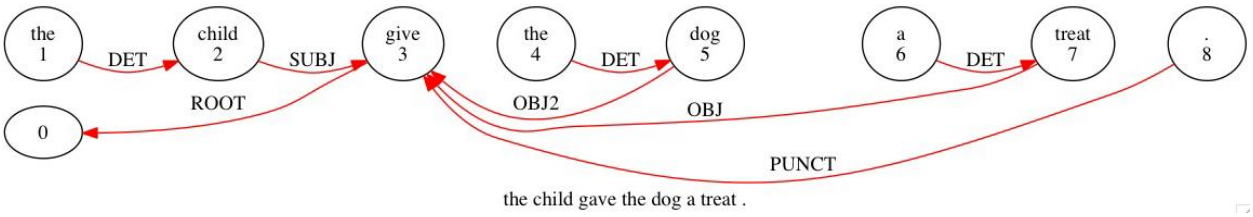
11.2 OBJ and OBJ2

OBJect identifies the direct object or first object of a verb. As with SUBJ, OBJ should be a nominal phrase. If it is a clause, it is COBJ. The head of OBJ is the verb for which it is acting as a direct object.



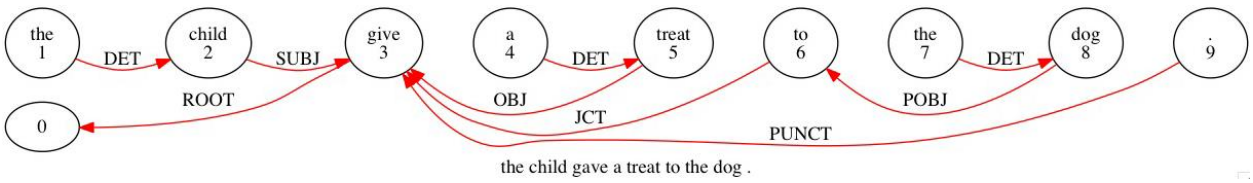
*PAR: the worker pushes the box .
%mor: art|the n|work&dv-AGT v|push-3S art|the n|box .
%gra: 1|2|DET 2|3|SUBJ 3|0|ROOT 4|5|DET 5|3|OBJ 6|3|PUNCT

OBJect2 identifies the indirect object or second object of a verb, if it is not introduced by a preposition. The head of OBJ2, like OBJ, is the verb for which it's acting as an indirect object. Here is an example with both OBJ and OBJ2:



*PAR: the child gave the dog a treat .
%mor: art|the n|child v|give-PAST art|the n|dog art|a n|treat .
%gra: 1|2|DET 2|3|SUBJ 3|0|ROOT 4|5|DET 5|3|OBJ2 6|7|DET 7|3|OBJ 8|3|PUNCT

The verb *gave* takes three arguments here. The SUBJ is *child*, the OBJ is *treat* and the OBJ2 is *dog*. OBJ2 is only used in case such as this where the beneficiary is not included in a prepositional phrase, as in *the child gave a treat to the dog.* In this latter case, *to the dog* is treated as an adjunct to the verb. Thus *to* ties as a JCT to *gave*, and *dog* ties as a POBJ to *to*.

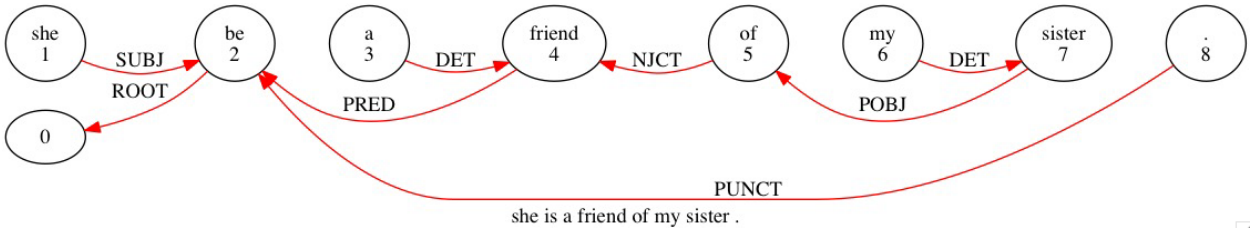


*PAR: the child gave a treat to the dog .
%mor: art|the n|child v|give-PAST art|a n|treat prep|to det|the n|dog .
%gra: 1|2|DET 2|3|SUBJ 3|0|ROOT 4|5|DET 5|3|OBJ 6|3|JCT 7|8|DET 8|6|POBJ 9|3|PUNCT

11.3 JCT, NJCT and POBJ

adJunCT = JCT identifies an adjunct that modifies a verb, adjective, or adverb (but not a noun – see section 4). The head of JCT is the verb, adjective, or adverb it modifies. This covers a very wide range of grammatical structures, but the word which is the dependent in the JCT relation is typically a preposition or adverb. Differentiating the type of JCT can be done by using searches that include the part of speech of the head on the %mor line. JCTs very commonly head up prepositional phrases. (A JCT may also be a noun in some cases, such as *way* in *she walked all the way home*.)

Nominal adJunCT = NJCT identifies an adjunct that modifies a noun rather than a verb, adjective, or adverb. Otherwise it behaves the same as JCT, able to take a POBJ and so on. The head of NJCT is the noun it modifies. Here is an example:



*PAR: she is a friend of my sister .
%mor: pro:sub|she cop|be&3S art|a n|friend prep|of pro:poss:det|my n|sister .
%gra: 1|2|SUBJ 2|0|ROOT 3|4|DET 4|2|PRED 5|4|NJCT 6|7|DET 7|5|POBJ 8|2|PUNCT

Prepositional OBJect = POBJ identifies the non-clausal nominal object of a preposition, such as *town* in the phrase *around town*. Clausal preposition objects are coded as CPOBJ. The head of POBJ is the preposition on which the nominal object depends.

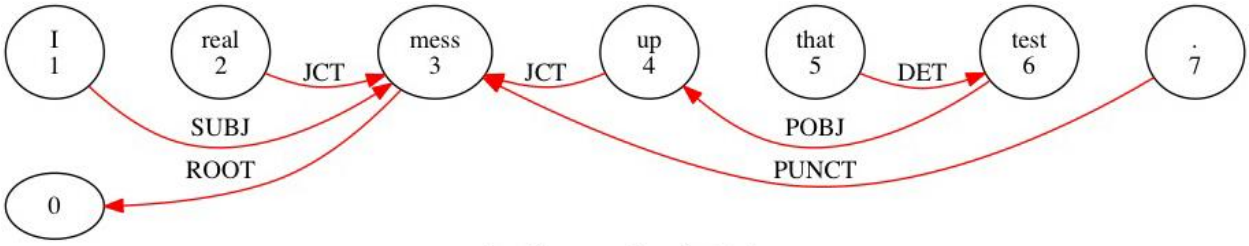
Note: JCT and POBJ also work together to describe phrasal verbs, despite the fact that their function is not truly adjunctive. For example, in the sentence *he took out the trash*, *out* is tied to *took* as a JCT and *trash* is tied to *out* as a POBJ.

This isn't a strictly accurate parse of the sentence, since *out the trash* is not an optional constituent here; rather, *the trash* would be more properly understood as an OBJ to the phrasal verb *took out*. However, because of the difficulties GRASP has in disambiguating these two cases, it is more consistent to treat this as a JCT-POBJ construction.

Note: When a prepositional phrase acts as the predicate in a *be* construction, treat it as a

JCT (and a POBJ, if there is one). A simple example would be *she is in the boathouse*.

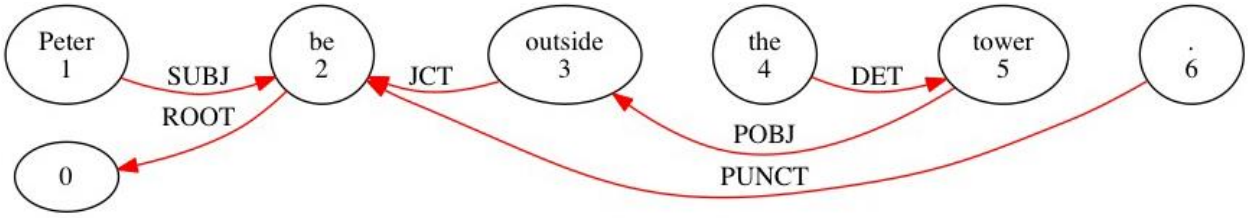
You might find it counterintuitive to tie *in* to *is* as a JCT there, since *in the boathouse* is, again, not an optional constituent. (As you'll see in the next section, it could be tempting to call *in the boathouse* a PRED.) It's more consistent to treat prepositional phrases this way, though, even when they're not adjunctive.



I really messed up that test.

*PAR: I really messed up that test.
%mor: pro:sub|I adv|real&dadj-LY v|mess-PAST prep|up det|that n|test .
%gra: 1|3|SUBJ 2|3|JCT 3|0|ROOT 4|3|JCT 5|6|DET 6|4|POBJ 7|3|PUNCT

First, note the adverb *really* tying as a JCT to the verb it modifies, *messed*. Otherwise, the purpose of this example is demonstrating how to diagram a phrasal verb in GRASP. As you can see, the way to handle the phrasal verb *messed up* is to treat *up that test* as an adjunct to *messed*, even though that's a slight compromise in accuracy.



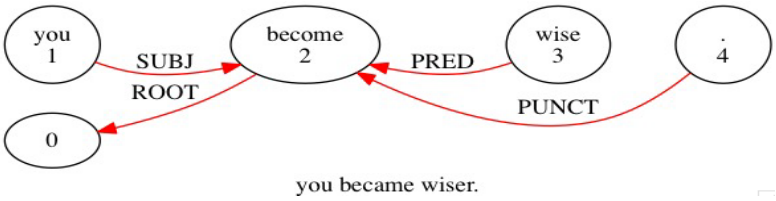
Peter is outside the tower .

*PAR: Peter is outside the tower .
%mor: n:prop|Peter cop|be&3S prep|outside art|the n|tower .
%gra: 1|2|SUBJ 2|0|ROOT 3|2|JCT 4|5|DET 5|3|POBJ 6|2|PUNCT

In this subject-*be*-PP construction. *outside the tower* is treated as a JCT to *is* in order to maximize consistency in the treatment of prepositional phrases, allowing for the fact that this leaves the copula *is* in the strange position of not appearing to take any predicate. Such structures can be searched by looking for copulas linked POBJ.

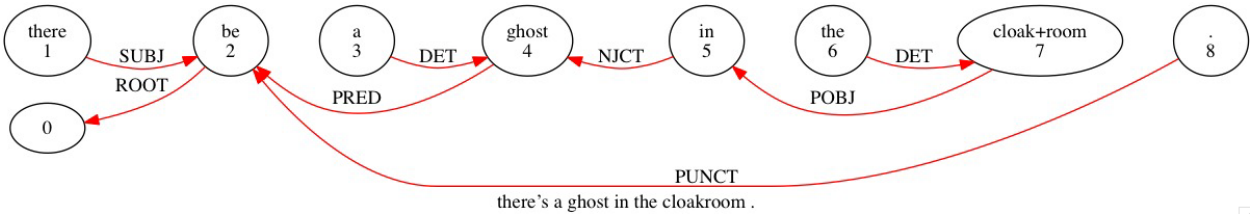
11.4 PRED

PREDicate identifies a nominal or adjectival predicate. This is usually an argument of the verb *be*, although other verbs such as *become* and *grow* (in the sense of *I grow weary of your complaints*) can also take a predicate. The head of PRED is the verb of which it is an argument. Here is a simple adjectival predicate.



*PAR: you became wiser .
%mor: pro:sub|you v|become-PAST adj|wise-CP .
%gra: 1|2|SUBJ 2|0|ROOT 3|2|PRED 4|2|PUNCT

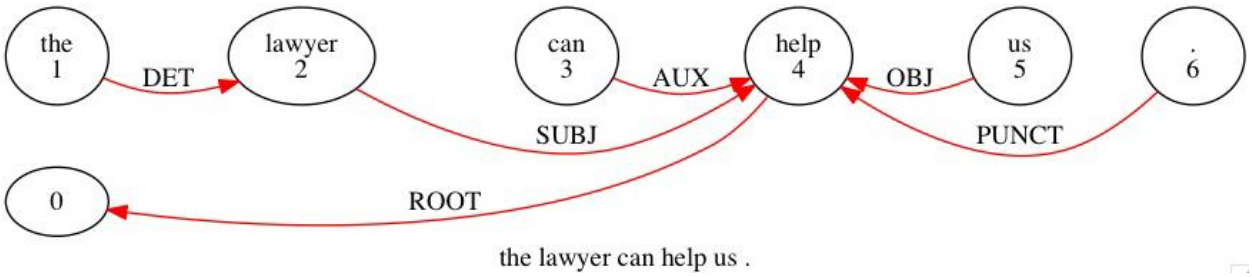
PRED also participates along with SUBJ in *there's-an-X* constructions. Treat *there* as an existential pronoun in these constructions, tying it to the *be* verb as a SUBJ. Then tie the introduced noun phrase to the *be* verb as a PRED. (Do the same for *here's-an-X*.) Here is an example:



*PAR: there's a ghost in the cloakroom .
%mor: pro:exist|there~cop|be&3S art|a n|ghost prep|in art|the n|+n|cloak+n|room .
%gra: 1|2|SUBJ 2|0|ROOT 3|4|DET 4|2|PRED 5|4|NJCT 6|7|DET 7|5|POBJ 8|2|PUNCT

11.5 AUX

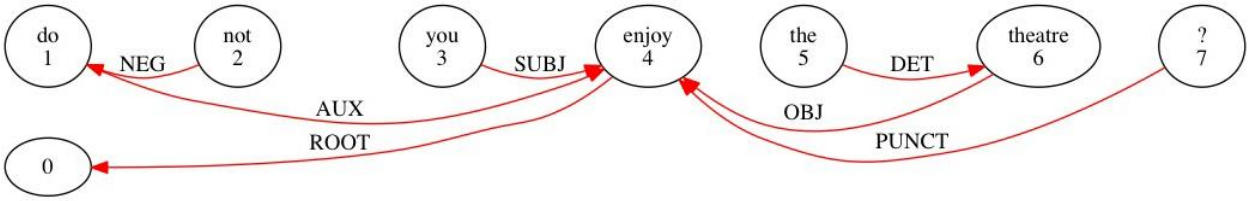
AUXiliary identifies an auxiliary or modal of a main verb, such as *can* or *should*. The head of AUX is the lexical verb. When multiple AUXes are present (*could have guessed*), each again ties individually to the main verb, for consistency. (None of the AUXes tie to each other.) Here is a simple auxiliary construction. Although *can* carries the tense information, the ROOT of the sentence is still the main verb, *help*. As mentioned above, *lawyer* ties directly to *help* as the SUBJ (it does not tie to *can*, the AUX).



*PAR: the lawyer can help us .
%mor: art|the n|lawyer mod|can v|help pro:obj|us .
%gra: 1|2|DET 2|4|SUBJ 3|4|AUX 4|0|ROOT 5|4|OBJ 6|4|PUNCT

11.6 NEG

NEGation identifies a verbal negator; i.e. *not* (including its contracted form *-n't*). In modern English, the presence of a NEG is nearly always accompanied by an AUX, a form of *be* as a main verb (*she isn't friendly*), or sometimes a participle (*consider not telling anyone that*). NEG can also in some cases negate a relativizer (see section 10 for details on this). The head of NEG is the AUX, verb, or relativizer it negates.



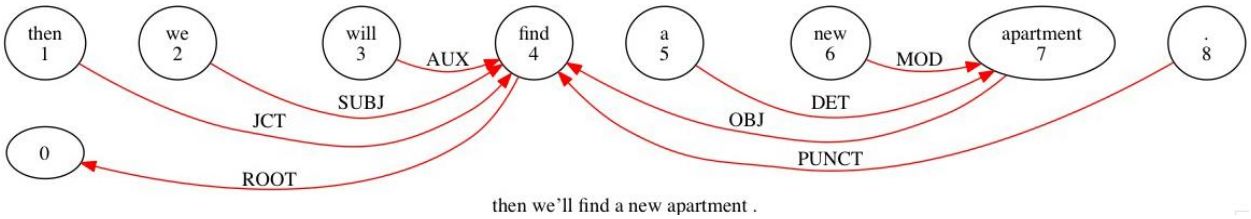
don't you enjoy the theatre ?

*PAR: don't you enjoy the theatre ?
%mor: mod|do~neg|not pro:sub|you v|enjoy art|the n|theatre ?
%gra: 1|4|AUX 2|1|NEG 3|4|SUBJ 4|0|ROOT 5|6|DET 6|4|OBJ 7|4|PUNCT

This demonstrates an AUX-leading question as well as a negated AUX. Structurally it's no different from *you don't enjoy the theatre*, just rearranged. The NEG ties to the AUX, not the main verb *enjoy*.

11.7 MOD and POSS

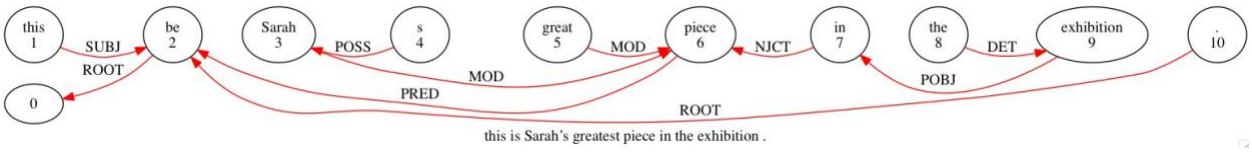
MODifier identifies a non-clausal modifier which precedes the noun it modifies. This is usually an adjective, a possessive, or another noun (as in *the cookie factory*, where *cookie* is a MOD to *factory*). The head of MOD is the noun being modified. When multiple MODs modify the same noun, each one ties to that noun separately. Here is a simple MOD construction. Both the DET *a* and the MOD *new* tie individually to *apartment* (*a* does not tie to *new*).



then we'll find a new apartment .

*PAR: then we'll find a new apartment .
%mor: adv:tem|then pro:sub|we~mod|will v|find art|a adj|new n|apartment .
%gra: 1|4|JCT 2|4|SUBJ 3|4|AUX 4|0|ROOT 5|7|DET 6|7|MOD 7|4|OBJ 8|4|PUNCT

POSSessive identifies the relation between the English possessive suffix and the noun it affixes to, such as *-'s* in *John's book*. The head of POSS is the noun to which it attaches (*John* in the given example, not *book*). The affixed noun then ties as a MOD to the noun it specifies (so *John* ties to *book*). In this next example, *-'s* ties to *Sarah* as a POSS, while *Sarah* ties to *piece* as a MOD (*greatest* also ties to *piece* as a separate MOD).



*PAR: this is Sarah's greatest piece in the exhibition .
%mor: pro:dem|this cop|be&3S n:prop|Sarah~poss|s adj|great&SP n|piece prep|in art|the n|exhibition .
%gra: 1|2|SUBJ 2|0|ROOT 3|6|MOD 4|3|POSS 5|6|MOD 6|2|PRED 7|6|NJCT 8|9|DET 9|7|POBJ 10|2|ROOT

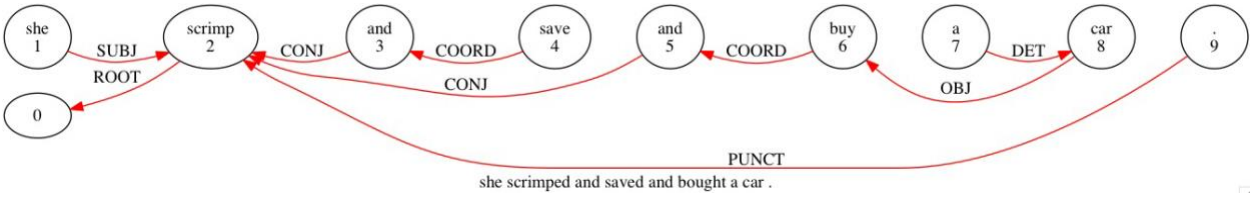
11.8 CONJ and COORD

The relations of CONJ and COORD only apply to non-clausal coordination or combination. These relations use the coordinators *and* and *or* ([scat coord]). The use of conjunctions ([scat conj]) to link subordinate clauses with main clauses is described by the **LINK** relation, discussed later. CONJ and COORD work together in the following way.

COORD first links the last element of a conjoined series to the coordinator.

CONJ then links the coordinator to the preceding element or enumerated element. For example, in the phrase *bold and eloquent woman*, first *eloquent* ties to *and* as a COORD, then *and* ties to *bold* as a CONJ, finally *bold* ties to *woman* as a MOD. The overall effect is to unite *bold and eloquent* into one constituent headed by its first element, *bold* which then modifies *woman*.

Here is an example of a sentence with two CONJ-COORD constructions. There are three conjoined verb phrases. The first one, *scrimped*, serves as the ROOT, while the other two, *saved* and *bought*, are first combined through COORD and then linked to the main verb through CONJ.



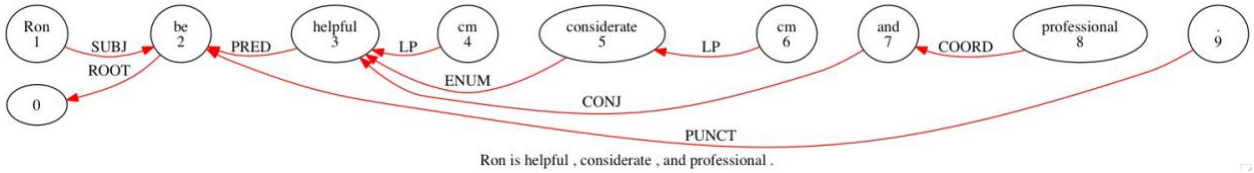
*PAR: she scrimped and saved and bought a car .
%mor: pro:sub|she v|scrimp-PAST conj|and v|save-PAST conj|and v|buy-PAST art|a n|car .
%gra: 1|2|SUBJ 2|0|ROOT 3|2|CONJ 4|3|COORD 5|2|CONJ 6|5|COORD 7|8|DET 8|6|OBJ 9|2|PUNCT

11.9 ENUM

ENUMeration joins a series of elements that are not linked through COORD. Each element is linked to the first item in the series which serves as the head. The series could

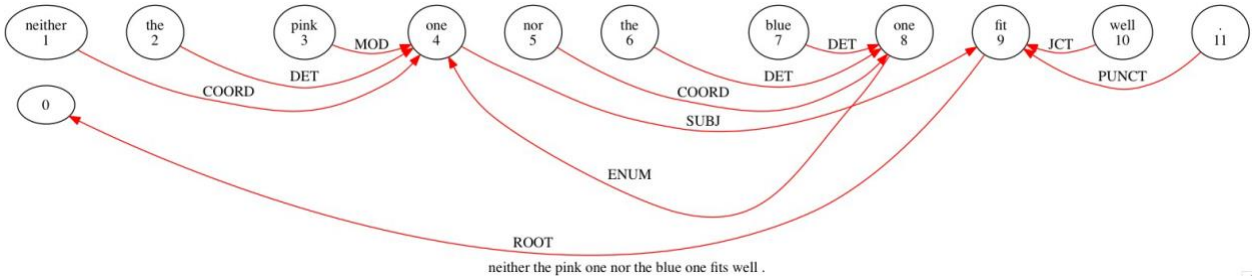
have any length, as occurs when children recite the letters of the alphabet or count to 20. In some cases the series will end with a coordinator and final element, but the items before the coordinator are still in an enumerated series, as in *we laughed, schmoozed, had a great time.*

Here is an example of a series of two items (*helpful, considerate*) with a final coordination (*and professional*):



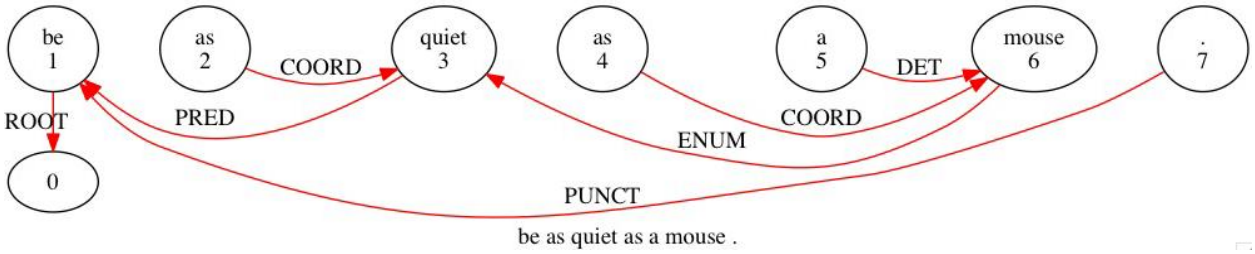
*PAR: Ron is helpful , considerate , and professional .
%mor: n:prop|Ron cop|be&3S adj|helpful cm|cm adj|considerate cm|cm conj|and adj|professional .
%gra: 1|2|SUBJ 2|0|ROOT 3|2|PRED 4|3|LP 5|3|ENUM 6|5|LP 7|3|CONJ 8|7|COORD 9|2|PUNCT

ENUM can also bind together more complex phrases as in the following example where it binds together *neither the pink one* and *nor the blue one* into one constituent headed by *[pink] one*, which can then act as the SUBJ of the sentence. Each of *neither* and *nor* is tied to its corresponding item as a COORD, and then *[blue] one* is tied to *[pink] one* as an ENUM.



*PAR: neither the pink one nor the blue one fits well .
%mor: conj|neither art|the adj|pink n|one conj|nor art|the adj|blue n|one v|fit-3S adv|well .
%gra: 1|4|COORD 2|4|DET 3|4|MOD 4|9|SUBJ 5|8|COORD 6|8|DET 7|8|DET 8|4|ENUM 9|0|ROOT 10|9|JCT 11|9|PUNCT

Here is another example in which the entire phrase *as quiet as a mouse* is bound into one constituent, with the PRED *quiet* as its head. This is accomplished by tying each *as* to its corresponding item as a COORD, and then tying *mouse* to *quiet* as an ENUM.

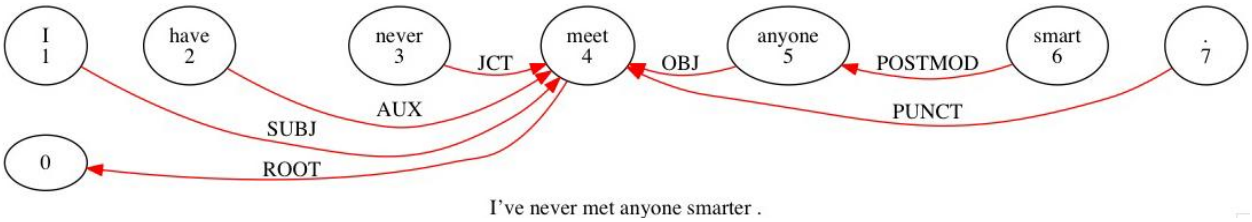


*PAR: be as quiet as a mouse .
%mor: v|be prep|as adj|quiet prep|as art|a n|mouse .

%gra: 1|0|ROOT 2|3|COORD 3|1|PRED 4|6|COORD 5|6|DET 6|3|ENUM 7|1|PUNCT

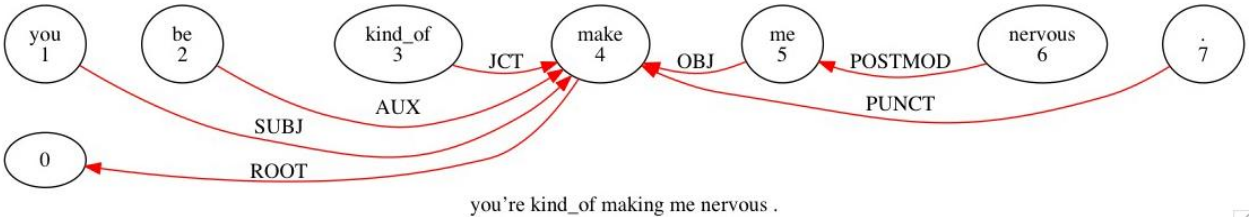
11.10 POSTMOD

POSTMODifier identifies a postposed nominal modifier, which might be either an adjective or another noun. POSTMOD can express resultative relations (*I painted the barn red*; *let's keep it a private affair*) and can also serve when a modifying phrase gravitates toward the end of the phrase due to size (*he dug a hole wider than a pickup truck*). There may be intervening content in the sentence (*she made it out of the haunted house alive*, where *alive* is a POSTMOD to *she*). As with MOD, the head of POSTMOD is the noun it modifies.



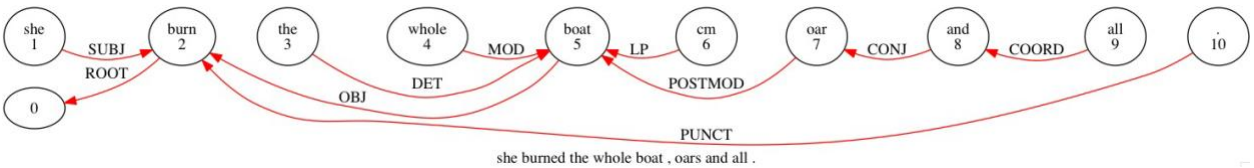
*PAR: I've never met anyone smarter .
%mor: pro:sub|I~aux|have adv|never part|meet&PASTP pro:indef|anyone adj|smart-CP .
%gra: 1|4|SUBJ 2|4|AUX 3|4|JCT 4|0|ROOT 5|4|OBJ 6|5|POSTMOD 7|4|PUNCT

This is a very basic instance of POSTMOD, where *smart* modifies *anyone* but is postposed.



*PAR: you're kind_of making me nervous .
%mor: pro:sub|you~aux|be&PRES adv|kind_of part|make-PRESP pro:obj|me adj|nervous .
%gra: 1|4|SUBJ 2|4|AUX 3|4|JCT 4|0|ROOT 5|4|OBJ 6|5|POSTMOD 7|4|PUNCT

making me nervous exhibits a resultative relationship. *me* ties as an OBJ to *making*, while *nervous*, the resultative complement, ties to *me* as a POSTMOD. Other verbs like *get*, *keep*, and *put* can participate in similar constructions (e.g. *get the crowd excited*, *keep it secret*, *put the blazer on*).



*PAR: she burned the whole boat , oars and all .
%mor: pro:sub|she v|burn-PAST art|the adj|whole n|boat cm|cm n|oar-PL conj|and pro:indef|all .
%gra: 1|2|SUBJ 2|0|ROOT 3|5|DET 4|5|MOD 5|2|OBJ 6|5|LP 7|5|POSTMOD 8|7|CONJ 9|8|COORD 10|2|PUNCT

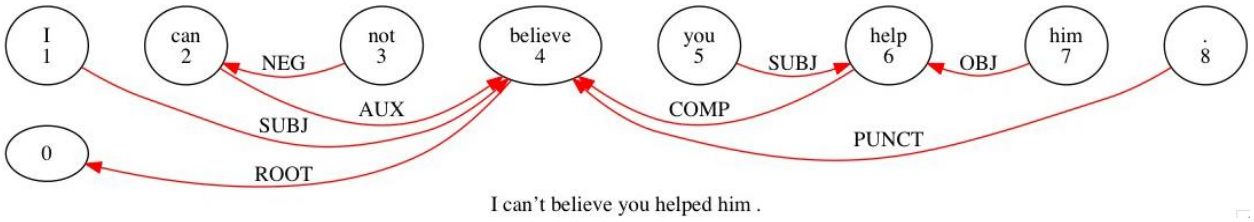
Here we see a detached phrase, *oars and all*, serving to add information to *boat* yet not explicitly connected to it by a verb or preposition. We'll solve this by tying *oar* to *boat* as a POSTMOD. *and all* then connect to *oar* using standard CONJ-COORD relations.

The trickiest question is where to tie the comma. Strictly speaking, this is not a list, but *the whole boat, oars and all* still gives the impression of being one large constituent in the sentence, so tying the comma all the way back to the root (as one would if the comma were separating distinct clauses) is unsatisfactory. Instead treat it as you would a list, tying the comma to the preceding *boat*, to preserve the constituency of *the whole boat, oars and all*.

POSTMOD can also be used in sentences such as *the woman leapt up, hat askew*. Here *hat askew* creates an unusual problem since it appears syntactically to belong to the sentence, yet it is not explicitly tied in with a preposition or a verb. Use the POSTMOD category to tie the "loose" NP to the preceding NP it describes. So *hat* should tie to *woman* as a POSTMOD. Don't confuse this with the appositive APP, where two NPs act as alternating names for the same entity.

11.11 COMP, LINK

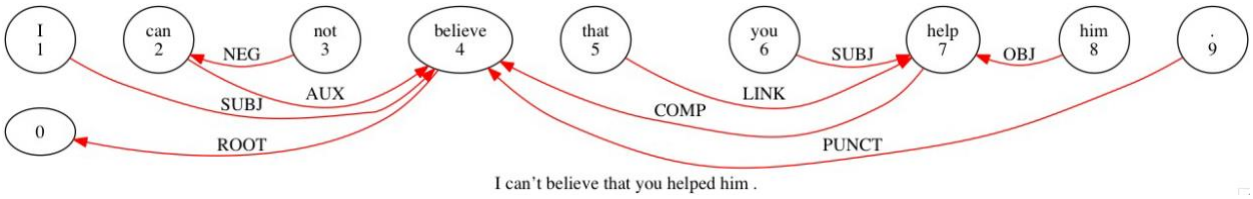
COMPlement identifies a finite clausal complement to a verb, such as a subordinate clause introduced by a verb like *say, know*, or *think* (*they know we're here*). The main verb of this subordinate clause, being its topmost element, is the item that should be classed as COMP. The head of COMP is the verb for which the clause is acting as a complement. Here is an example of COMP without LINK:



*PAR: I can't believe you helped him .
%mor: pro:sub|I mod|can~neg|not v|believe pro:subj|you v|help-PAST pro:obj|him .
%gra: 1|4|SUBJ 2|4|AUX 3|2|NEG 4|0|ROOT 5|6|SUBJ 6|4|COMP 7|6|OBJ 8|4|PUNCT

LINK identifies a word in a number of categories that can introduce a subordinate clause, such as complementizers (*that* in *I heard that you'd given it up*), relativizers (*who* in *the girl who told me about the concert*), and subordinate conjunctions (*unless* in *we're fine unless the power goes out*). One of the uses of LINK is to include the complementizer in

a COMP clause, as in this example, which adds *that* to the previous example:



*PAR: I can't believe that you helped him .
%mor: pro:sub|I mod|can~neg|not v|believe rel|that pro:subj|you v|help-PAST pro:obj|him .
%gra: 1|4|SUBJ 2|4|AUX 3|2|NEG 4|0|ROOT 5|7|LINK 6|7|SUBJ 7|4|COMP 8|7|OBJ 9|4|PUNCT

LINK is also used to join a subordinating conjunction to the verb of a subordinate clause.
*** example missing ***

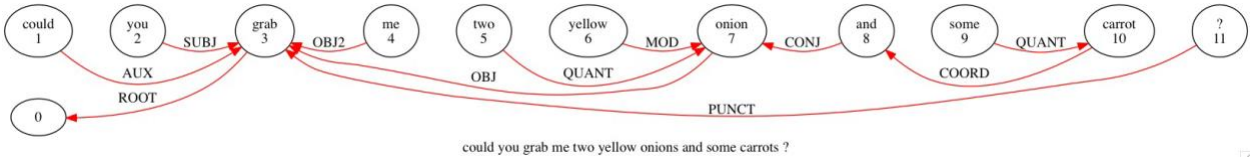
Sometimes such clauses can appear without main clauses. For example, an English speaker may say *unless the power goes out* on its own, in which case *unless* is still classed as a LINK. This is especially common with sentences beginning with the coordinators *and*, *but*, *or*, etc. In all these cases, the head of LINK is the main verb of its clause.

11.12 QUANT and PQ

QUANTifier identifies a nominal quantifier, such as *many* or *several*. Numbers that quantify nouns also fall into this category (*three foxes*), as do distributive terms like *each* and *any*. However, in all these cases, the QUANT must be modifying a noun phrase; if there is no noun phrase being modified, then the quantifier is presumably acting as a pronoun and should be classified as such. (For example, in *some say he's a traitor*, *some* is the SUBJ, not a QUANT.)

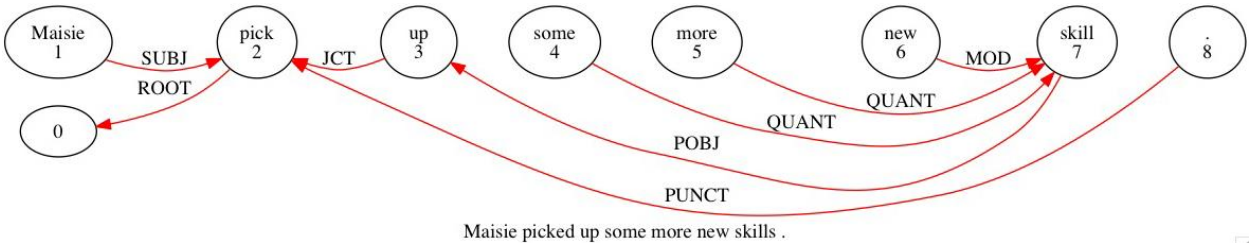
Like MODs, QUANTs do not stack when they quantify the same noun phrase, nor do they interact with MODs or a DET. All tie individually to the head noun. The head of QUANT is the noun it modifies.

Here is an example with two cases of QUANTs modifying noun phrases: *two* and *some*. Just like DET, QUANT doesn't interact with MOD at all, but instead ties separately to the noun it quantifies, while MOD does the same. So *two* ties to *onions* here, not *yellow*.



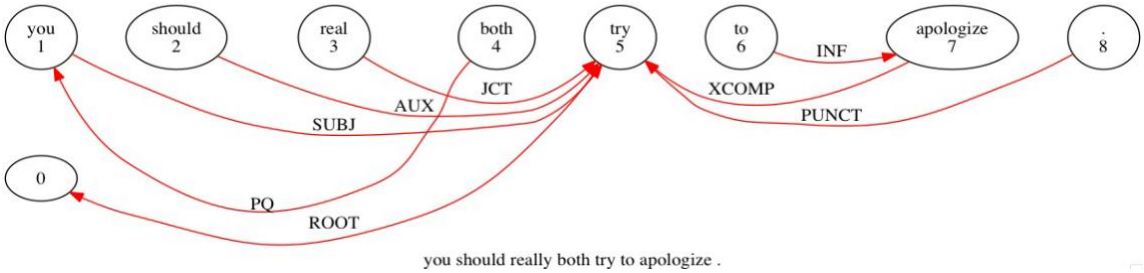
*PAR: could you grab me two yellow onions and some carrots ?
%mor: mod|could pro:sub|you v|grab pro:obj|me det:num|two adj|yellow n|onion-PL conj|and qn|some n|carrot-PL ?
%gra: 1|3|AUX 2|3|SUBJ 3|0|ROOT 4|3|OBJ2 5|7|QUANT 6|7|MOD 7|3|OBJ 8|7|CONJ 9|10|QUANT 10|8|COORD 11|3|PUNCT

Here is an example with two QUANTs and MOD linked to a single head:



*PAR: Maisie picked up some more new skills .
%mor: n:prop|Maisie v|pick-PAST prep|up qn|some qn|more adj|new n|skill-PL .
%gra: 1|2|SUBJ 2|0|ROOT 3|2|JCT 4|7|QUANT 5|7|QUANT 6|7|MOD 7|3|POBJ 8|2|PUNCT

PostQuantifier = PQ identifies a quantifier that follows rather than precedes its noun phrase. This mostly occurs with *both* (*we both tried it*) and *all* (*they all gave up*). As with POSTMOD, it's acceptable for some sentential content to intervene between the quantified noun and the PQ. The head of PQ is the noun it modifies.



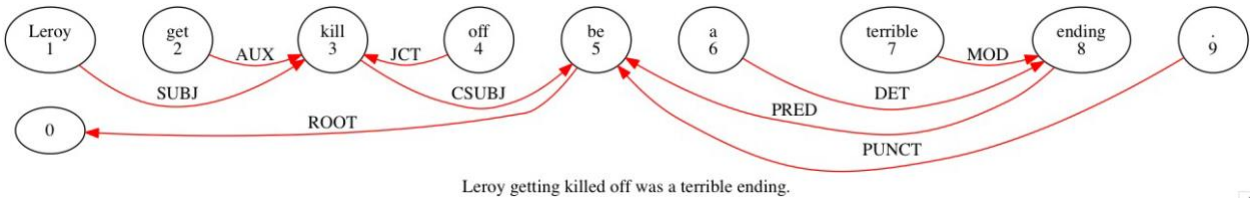
*PAR: you should really both try to apologize .
%mor: pro:sub|you mod|should adv|real&dadj-LY qn|both v|try inf|to v|apologize .
%gra: 1|5|SUBJ 2|5|AUX 3|5|JCT 4|1|PQ 5|0|ROOT 6|7|INF 7|5|COMP 8|5|PUNCT

It is not a problem that *should* and *really* intervene between *you* and *both* here. *both* still ties to *you* as a PQ, since that's the term it serves to quantify.

11.13 CSUBJ, COBJ, CPOBJ, CPRED

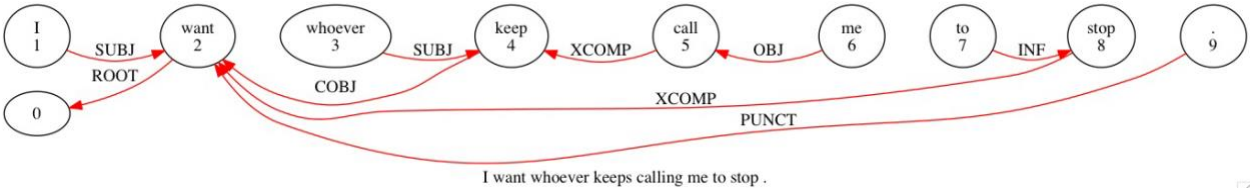
Clausal SUBJect = CSUBJ, Clausal OBJect = COBJ, Clausal Prepositional OBJect = CPOBJ, and Clausal PRED = CPRED are variations on the categories SUBJ, OBJ, POBJ, and PRED. In each of these cases, the appropriate syntactic role is being performed by an entire verbal clause (which could be either finite or non-finite) rather than a noun phrase.

CSUBJ: The main verb of that clause, being the topmost element, is the one that will be identified as CSUBJ, etc. From there, the clause ties to the rest of the sentence exactly as its non-clausal equivalent would. So, the head of CSUBJ, for example, is the verb for which that clause acts as the subject. In this next example, the entire clause *Leroy getting killed off* is the subject which is ascribed the quality of being *a terrible ending*. Thus the main verb of that clause, *killed*, is tied as a CSUBJ to *was*.



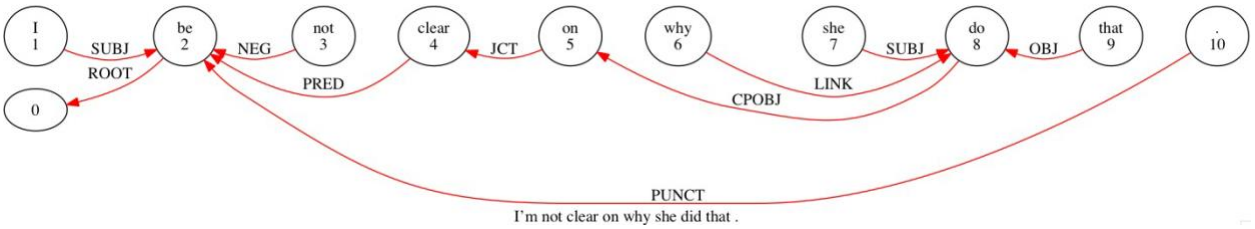
*PAR: Leroy getting killed off was a terrible ending.
%mor: n:prop|Leroy part|get-PRESP part|kill-PASTP prep|off cop|be&PAST&13S art|a adj|terrible n|ending .
%gra: 1|3|SUBJ 2|3|AUX 3|5|CSUBJ 4|3|JCT 5|0|ROOT 6|8|DET 7|8|MOD 8|5|PRED 9|5|PUNCT

COBJ: In this next example, *whoever keeps calling me* is a clause here acting as the object of *want*. So the main verb of this clause, *keeps*, ties to *want* as a COBJ.



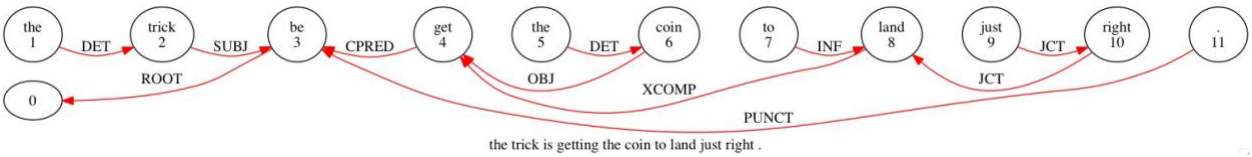
*PAR: I want whoever keeps calling me to stop .
%mor: pro:sub|I v|want pro:wh|whoever v|keep-3S part|call-PRESP pro:obj|me inf|to v|stop .
%gra: 1|2|SUBJ 2|0|ROOT 3|4|SUBJ 4|2|COBJ 5|4|COMP 6|5|OBJ 7|8|INF 8|2|COMP 9|2|PUNCT

CPOBJ: In this example, the clause *why she did that* acts as the object of the preposition *on*; thus its main verb *did* ties to *on* as a CPOBJ:



*PAR: I'm not clear on why she did that .
%mor: pro:sub|I~cop|be&1S neg|not adj|clear prep|on adv:wh|why pro:sub|she v|do-PAST pro:dem|that .
%gra: 1|2|SUBJ 2|0|ROOT 3|2|NEG 4|2|PRED 5|4|JCT 6|8|LINK 7|8|SUBJ 8|5|CPOBJ 9|8|OBJ 10|2|PUNCT

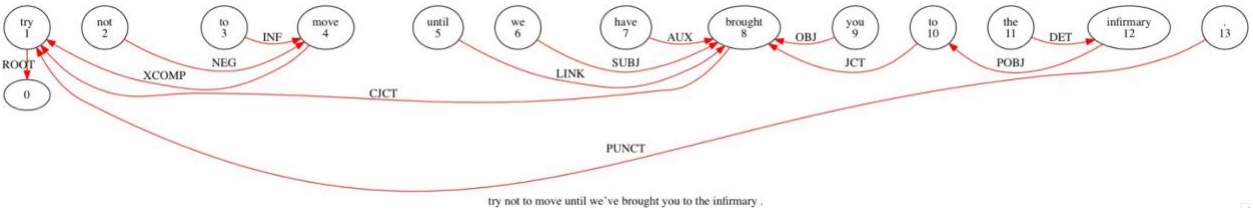
CPRED: In this example, *getting the coin to land just right* is the predicate to the main verb *be*. Thus, its main verb *getting* ties to *be* as a CPRED.



*PAR: the trick is getting the coin to land just right .
%mor: art|the n|trick cop|be&3S part|get-PRESP art|the n|coin inf|to v|land adv|just adv|right .
%gra: 1|2|DET 2|3|SUBJ 3|0|ROOT 4|3|PRED 5|6|DET 6|4|OBJ 7|8|INF 8|4|COMP 9|10|JCT 10|8|JCT 11|3|PUNCT

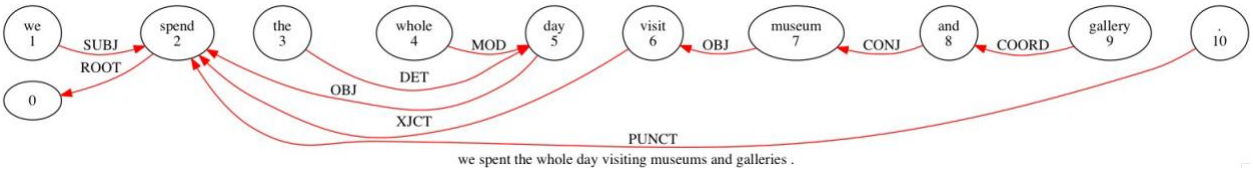
11.14 CJCT and XJCT

Clausal conJunCT = CJCT identifies a finite clause that functions as an adjunct to a verb, adjective, or adverb. The most common attachment is for a subordinate clause attached to a main clause. For example, in the sentence *you'll do it, because I said so*, the subordinate clause *because I said so* is a clausal conjunct on the verbof the main clause *do*.As with the clausal categories introduced in the previous section, the main verb within the adjoining clause (*said* in the above example) is the item that should be tagged CJCT, and its head is the main verb of the clause to which it adjoins (*do*). The conjunction *because* is linked to *said* through the LINK relation. Here is another example in which *until we have brought you to the infirmary* is a finite clausal adjunct, and so its main verb *brought* ties to *try* as a CJCT, and *until* functions as a LINK introducing the adjoining clause.



*PAR: try not to move until we've brought you to the infirmary .
%mor: v|try neg|not inf|to v|move conj|until pro:sub|we~aux|have part|brought-PASTP pro:obj|you prep|to art|the n|infirmary .
%gra: 1|0|ROOT 2|4|NEG 3|4|INF 4|1|COMP 5|8|LINK 6|8|SUBJ 7|8|AUX 8|1|CJCT 9|8|OBJ 10|8|JCT 11|12|DET 12|10|POBJ 13|1|PUNCT

XadJunCT = XJCT identifies a non-finite clause (that is, generally, one headed by an infinitive verb or participle) that otherwise behaves like a CJCT, as in *to build a ship, first gather some wood*. Again, the main verb in the adjoining clause (*build* in the example) is tagged as XJCT and its head is the main verb of the clause to which it adjoins (*gather*). In the following example, *visiting museums and galleries* is a non-finite clausal adjunct whose main verb ties to *spend* as an XJCT.



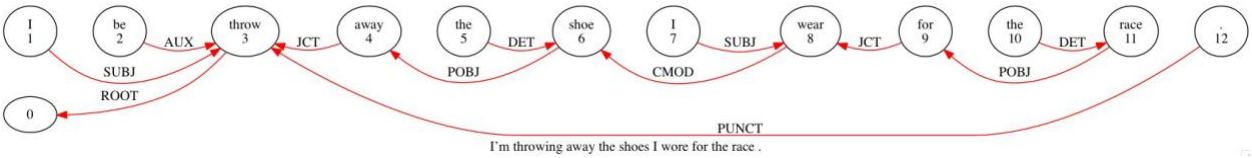
*PAR: we spent the whole day visiting museums and galleries .
%mor: pro:sub|we v|spend-PAST art|the adj|whole n|day part|visit-PRESP n|museum-PL

conj|and n|gallery-PL .
%gra: 1|2|SUBJ 2|0|ROOT 3|5|DET 4|5|MOD 5|2|OBJ 6|2|XJCT 7|6|OBJ 8|7|CONJ 9|8|COORD 10|2|PUNCT

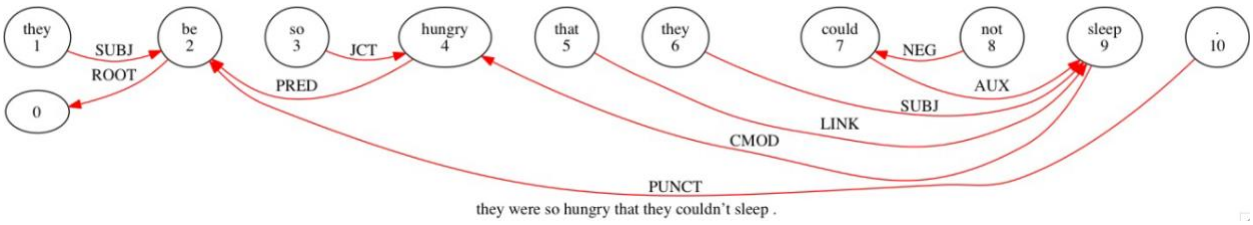
11.15 CMOD and XMOD

Clausal MODifier = CMOD identifies a finite clause that modifies a noun, as in *the boy (that) Tricia took to prom*. CMOD also sometimes modifies an adjective or adverb, if that adjective/adverb takes a complement (*we're happy that you made it*) or participates in a resultative expression (*so quietly I couldn't hear them*). CMOD functions much like CJCT and XJCT: the main verb within the modifier clause is the item tagged CMOD, and its head is the noun, adjective, or adverb it modifies.

In the following example, the clausal adjunct *I wore for the race* does not modify the main verb *throwing*, but rather *shoes* – this clause specifies a particular pair of the speaker's shoes. Therefore *wore* ties to *shoes* as a CMOD rather than tying to *throwing* as a CJCT.

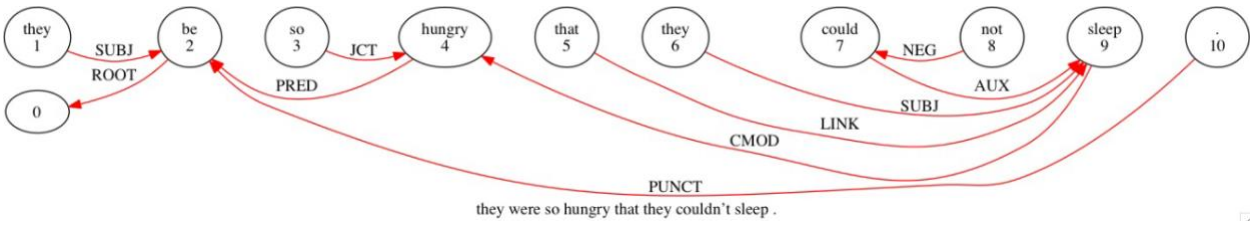


*PAR: I'm throwing away the shoes I wore for the race .
%mor: pro:sub|I~aux|be&1S part|throw-PRESP prep|away art|the n|shoe-PL pro:sub|I v|wear-PAST prep|for art|the n|race .
%gra: 1|3|SUBJ 2|3|AUX 3|0|ROOT 4|3|JCT 5|6|DET 6|4|POBJ 7|8|SUBJ 8|6|CMOD 9|8|JCT 10|11|DET 11|9|POBJ 12|3|PUNCT



*PAR: they were so hungry that they couldn't sleep .
%mor: pro:sub|they cop|be&PAST adv|so adj|hungry rel|that pro:sub|they mod|could~neg|not v|sleep .
%gra: 1|2|SUBJ 2|0|ROOT 3|4|JCT 4|2|PRED 5|9|LINK 6|9|SUBJ 7|9|AUX 8|7|NEG 9|4|CMOD 10|2|PUNCT

CMOD can also modify an adjective, as in this example:

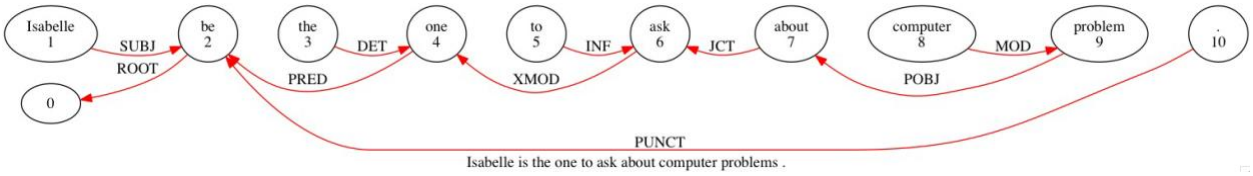


*PAR: they were so hungry that they couldn't sleep .
%mor: pro:sub|they cop|be&PAST adv|so adj|hungry rel|that pro:sub|they mod|could~neg|not

v|sleep .
%gra: 1|2|SUBJ 2|0|ROOT 3|4|JCT 4|2|PRED 5|9|LINK 6|9|SUBJ 7|9|AUX 8|7|NEG 9|4|CMOD 10|2|PUNCT

XMODifier identifies a non-finite clause that otherwise behaves like a CMOD, as in *the person running the store*; *a good day to take a walk*; *she's eager to teach them*.

Once again, the main verb within the modifying clause is the one tagged XMOD, and its head is the noun, adjective, or adverb it modifies. In this example, the head is *one*.



*PAR: Isabelle is the one to ask about computer problems .
%mor: n:prop|Isabelle cop|be&3S art|the n|one inf|to v|ask prep|about n|computer n|problem-PL .
%gra: 1|2|SUBJ 2|0|ROOT 3|4|DET 4|2|PRED 5|6|INF 6|4|XMOD 7|6|JCT 8|9|MOD 9|7|POBJ 10|2|PUNCT

11.16 BEG, BEGP, END, ENDP

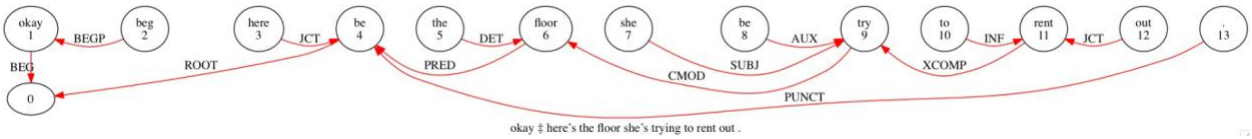
BEGinning identifies a clause-external element occurring at the beginning of an utterance. This might be, for example, an interjection like *hey* or *okay*, or a vocative like the addressee's name.

The head of BEG is 0, the LeftWall.

BEGinning Punctuation = BEGP identifies the double dagger ‡ which is employed to partition off a BEG element from the rest of the sentence for the purposes of MOR. The double dagger should always follow a BEG.

The head of BEGP is the BEG it follows.

Here is an example of BEG and BEGP in which the BEG *okay* ties directly to 0, while the BEGP ties to *okay*.



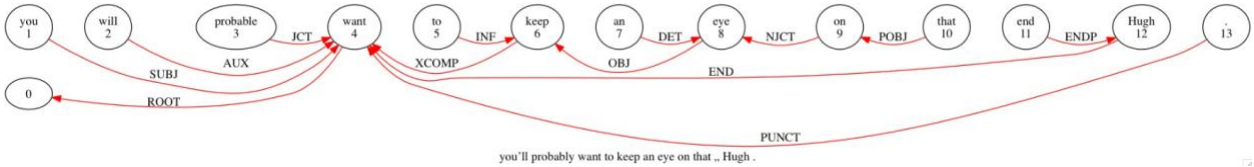
*PAR: okay ‡ here's the floor she's trying to rent out .
%mor: co|okay beg|beg pro:exist|here~cop|be&3S art|the n|floor pro:sub|she~aux|be&3S part|try-PRESP inf|to v|rent prep|out .
%gra: 1|0|BEG 2|1|BEGP 3|4|JCT 4|0|ROOT 5|6|DET 6|4|PRED 7|9|SUBJ 8|9|AUX 9|6|CMOD 10|11|INF 11|9|COMP 12|11|JCT 13|4|PUNCT

END identifies a clause-external element occuring at the end of an utterance. This includes interjections, vocatives, and other elements similar to those that can act as BEGs. However, it doesn't include tag questions like *...don't you?* Whereas the head of

BEG is 0, the head of END is the ROOT.

END Punctuation = ENDP identifies the double comma „ which is employed to partition off an END element much like the double dagger. The double comma should always precede an END or a TAG. The head of ENDP is the END it precedes.

Here is an example of END and ENDP in which the END here *Hugh* ties to the ROOT *want* (then the ENDP ties to *Hugh*) and the ENDP ties to END.

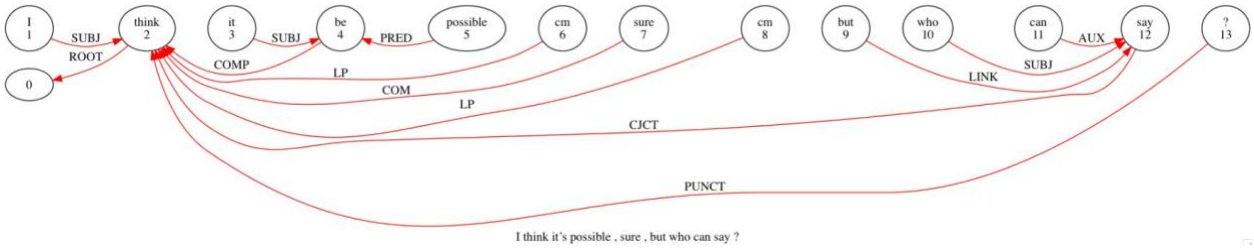


*PAR: you'll probably want to keep an eye on that „ Hugh .
%mor: pro:sub|you~mod|will adv|probable&dadj-LY v|want inf|to v|keep art|an n|eye prep|on pro:dem|that end|end n:prop|Hugh .
%gra: 1|4|SUBJ 2|4|AUX 3|4|JCT 4|0|ROOT 5|6|INF 6|4|COMP 7|8|DET 8|6|OBJ 9|8|NJCT 10|9|POBJ 11|12|ENDP 12|4|END 13|4|PUNCT

11.17 COM and TAG

COMmunicator identifies any clause-external element occuring medially rather than initially or finally. Depending on the transcriber, such an element might be partitioned off with commas (*I walked into the courtroom, right, and here's this guy telling me…*), or it might not be partitioned at all (*I know you don't like it Li but it's the best solution we've got*). Like END, and unlike BEG, the head of COM is the ROOT.

Here is an example:



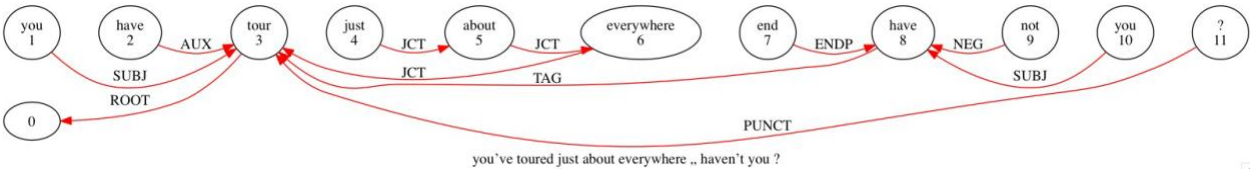
*PAR: I think it's possible , sure , but who can say ?
%mor: pro:sub|I v|think pro|it~cop|be&3S adj|possible cm|cm co|sure cm|cm conj|but pro:wh|who mod|can v|say ?
%gra: 1|2|SUBJ 2|0|ROOT 3|4|SUBJ 4|2|COMP 5|4|PRED 6|2|LP 7|2|COM 8|2|LP 9|12|LINK 10|12|SUBJ 11|12|AUX 12|2|CJCT 13|2|PUNCT

Both of the commas used to partition off *sure* are also tied to the ROOT (since they're not delineating items in a list, but rather entire clauses).

TAG identifies the verb of a tag question at the end of a sentence (*…doesn't it?*). As mentioned in the previous section, this is not considered an END, but has a category of its

own since it takes a more complex set of syntactic interactions. However, as with END, there may be a double comma preceding the tag verb; if so, tie it to the tag verb as an ENDP. The head of TAG, like END, is the ROOT.

Here is an example of TAG in action. The tag verb, *have*, ties to the ROOT *toured*. That tag verb takes its own NEG *not* and even its own SUBJ *you*, so it's more complex than an END. However, just as if it were an END, the double comma ties to it as an ENDP. PUNCT still ties to the ROOT, because we want to tie PUNCT consistently to the ROOT in all cases.

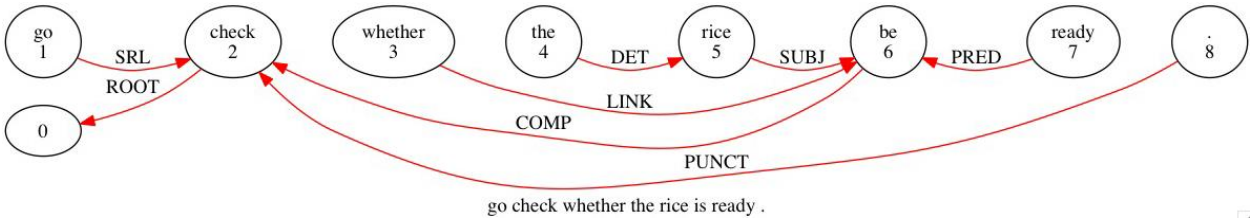


*PAR: you've toured just about everywhere „ haven't you ?
%mor: pro:sub|you~aux|have part|tour-PASTP adv|just adv|about adv|everywhere end|end aux|have~neg|not pro:sub|you ?
%gra: 1|3|SUBJ 2|3|AUX 3|0|ROOT 4|5|JCT 5|6|JCT 6|3|JCT 7|8|ENDP 8|3|TAG 9|8|NEG 10|8|SUBJ 11|3|PUNCT

11.18 SRL, APP

SeRiaL identifies serial verbs, such as *come play* and *go see*. In each of these pairs, the second verb is treated as the head and the first verb is treated as the dependent. Thus, the first verb is tagged SRL and its head is the second verb.

Generally serial verb pairs are both bare verbs, but there's at least one exception: *get* in the sense of *get going* or another *get X-ing*. Notice that the second verb doesn't seem to relate to *get* as an object or complement, so the function of *get* appears more serial. (Contrast the more transparent *start packing*, *try spinning*, etc., where the second verb would tie as an COMP to the first verb.) Here is an example in which *go* ties to *check* as a SRL.



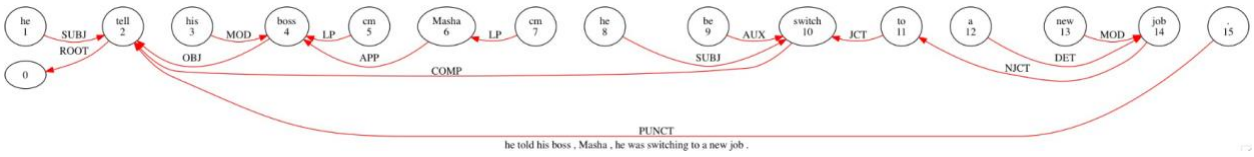
*PAR: go check whether the rice is ready .
%mor: v|go v|check conj|whether art|the n|rice cop|be&3S adj|ready .
%gra: 1|2|SRL 2|0|ROOT 3|6|LINK 4|5|DET 5|6|SUBJ 6|2|COMP 7|6|PRED 8|2|PUNCT

APPositive identifies an appositive noun or phrase, as in *Giles the baker*. The head of APP is the noun it corresponds to. However, which of the two apposed terms should be considered the head and which the APP will depend on the structure of the sentence.

In most straightforward constructions (as in the one above), the first term is the head

and the following term is the APP (so *baker* will tie to *Giles* as an APP). On the other hand, a sentence might be structured such that the second term – especially if it's a pronoun – will read more like the SUBJ (etc.) and the first term will read more like the APP. In the sentence *myself, I prefer provolone*, treat *I* as the SUBJ and tie *myself* to it as an APP.

Remember the essential quality of APP: both noun phrases denote the same entity, and serve as two different ways of naming it. That makes APP unlike our *hat askew* example from back in section 9, which is why APP is inappropriate there, and POSTMOD is used instead.



*PAR: he told his boss , Masha , he was switching to a new job .
%mor: pro:sub|he v|tell-PAST pro:poss:det|his n|boss cm|cm n:prop|Masha cm|cm pro:sub|he aux|be&PAST&13S part|switch-PRESP prep|to art|a adj|new n|job .
%gra: 1|2|SUBJ 2|0|ROOT 3|4|MOD 4|2|OBJ 5|4|LP 6|4|APP 7|6|LP 8|10|SUBJ 9|10|AUX 10|2|COMP 11|10|JCT 12|14|DET 13|14|MOD 14|11|NJCT 15|2|PUNCT

Here *Masha* ties to *boss* as an APP. Notice that the transcriber put commas here to partition *Masha* from the rest of the sentence. This is optional – it makes no difference in how *Masha* ties to *boss* – but take note of how the commas are treated, each one tied to the word preceding. Essentially the commas behave exactly as if this were a list of items conjoined with ENUM (so they tie directly to those items, rather than the ROOT).

11.19 NAME, DATE

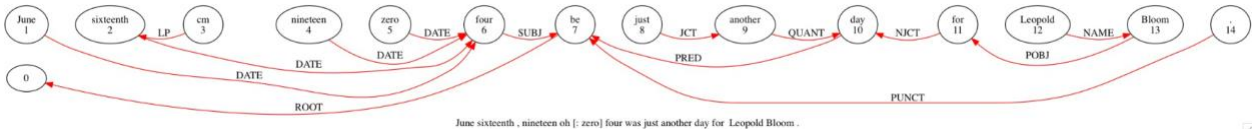
NAME identifies relations within a string of elements that are functioning together to act as one name, like *Deandre Clarks* and *South Carolina*. When transcribing, these should generally be conjoined with underscores to form single syntactic units (*Deandre_Clarks* and *South_Carolina*). However, if the two or more elements of a name are not or cannot be conjoined (for instance, if there's intervening material: *Deandre &uh Clarks*), the NAME tag will be required.

Use NAME to tie every non-final element in the name to the final element. Thus in *South Carolina*, *South* will tie to *Carolina* as a NAME.

DATE identifies relations within a string of elements that are functioning together to act as a date, like *July twentieth nineteen ninety*. Unlike proper names, these are never conjoined with underscores to form single syntactic units, since the combination of numbers would be endless. Thus DATE is required in all cases.

For consistency with NAME, tie every non-final element in the date to the final element. Thus in *July twentieth nineteen ninety*, all elements but *ninety* will tie to it as DATEs.

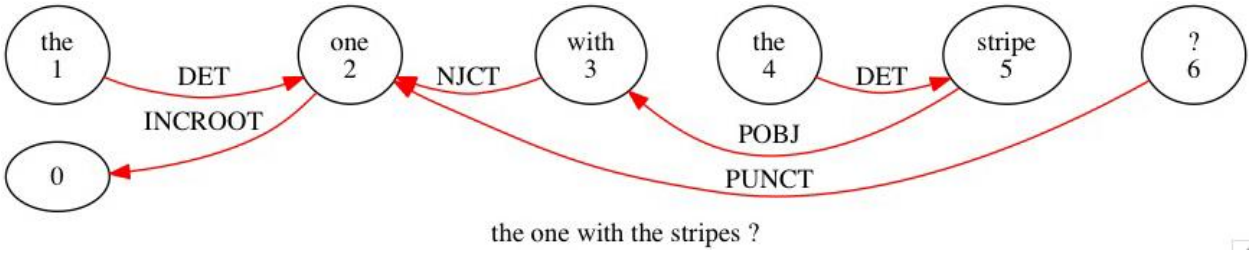
\



*PAR: June sixteenth , nineteen oh [: zero] four was just another day for Leopold Bloom .
%mor: n:prop|June adj|sixteenth cm|cm det:num|nineteen det:num|zero det:num|four cop|be&PAST&13S adv|just qn|another n|day prep|for n:prop|Leopold n:prop|Bloom .
%gra: 1|6|DATE 2|6|DATE 3|2|LP 4|6|DATE 5|6|DATE 6|7|SUBJ 7|0|ROOT 8|9|JCT 9|10|QUANT 10|7|PRED 11|10|NJCT 12|13|NAME 13|11|POBJ 14|7|PUNCT

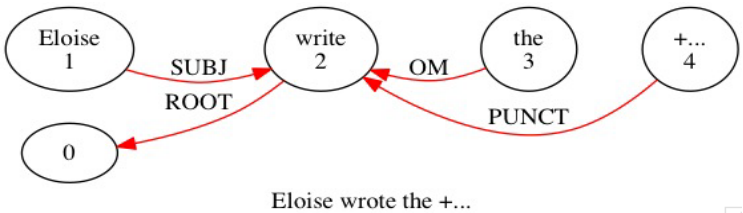
11.20 INCROOT, OM

INCROOT is a catch-all category for any word that must substitute in the role of ROOT when no proper ROOT is available. This may be because the sentence lacks a verb altogether, or because any verb present is not top-level (say, it's part of a clause depending on another phrase). When no ROOT is available, choose the topmost word in the sentence and name it the INCROOT. The head of INCROOT is 0. Here is an example of INCROOT in action. Since there's no verb in this sentence, there is no proper ROOT, and we must assign the label of INCROOT to another element in the sentence. The topmost existing element is *one*, upon which the determiner *the* depends, as well as the entire NJCT phrase that follows. Thus, it is the logical choice to be identified as the INCROOT. Naturally, PUNCT will now tie to *one* as if it were the ROOT.



*PAR: the one with the stripes ?
%mor: art|the n|one prep|with art|the n|stripe-PL ?
%gra: 1|2|DET 2|0|INCROOT 3|2|NJCT 4|5|DET 5|3|POBJ 6|2|PUNCT

OMission is another catch-all for any determiner, modifier, or other dependent word which is left headless due to an omission, such as the speaker trailing off or being interrupted and hence leaving their sentence unfinished. The head of OM is whatever the head of its head would have been. (So the head of OM is the ROOT, for instance, if what's been omitted was the OBJ of the root verb.)



*PAR: Eloise wrote the +...
%mor: n:prop|Eloise v|write-PAST art|the +...
%gra: 1|2|SUBJ 2|0|ROOT 3|2|OM 4|2|PUNCT

12 GRs for other languages

Although many of the basic GRs for English can be found in other languages, there are often relations in these other languages that are not found in English.

12.1 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.

12.2 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 the exception of INF.

2. Also, Chinese does not have a finite, non-finite distinction on verbs. This makes the "X" relations of English irrelevant and only CSUBJ, COMP, CMOD, PRED, and CJCT are needed. The main verb is the head of the subordinate clause.

3. 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.

4. **SFP** is a relation between the sentence final particle and "0".

5. **SRL** is for serial verb constructions. For Chinese, unlike English, the head of SRL is the first verb, not the second.

6. **POSS** in Chinese is the relation 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.

7. **JCT** can be used in Chinese for classifier phrases that such as *yi1 bian4*.

8. **JCT** also extends also to the complements of directional verbs, as in this example:
   他 跑 进 瓶子 里面.
   pro|ta1=he v|pao3=run v:dirc|jin4=enter n|ping2-NOM=bottle post|li3mian4=inside.
   1|2|SUBJ 2|0|ROOT 3|2|SRL 4|5|POSTO 5|2|JCT 6|2|PUNCT
   Note that the JCT is attached to the first of the two serial verbs which is the head of the serial verb pair.

9. **CJCT** is used in cases where there are no subordinating conjunctions, as in this example:
   青蛙 一 看 就 没 了 .
   n|qing1wa1=frog adv|yi1=just_as v|kan4=look adv|jiu4=just neg|mei2=not asp|le .
   1|6|COM 2|4|SUBJ 3|4|JCT 4|0|ROOT 5|6|JCT 6|4|PRED 7|6|CLEFT 17 8|4|PUNCT
   It can also extend to linking the phrase mei2 you3 to the main verb.
   没 有 讲 完.
   neg|mei2=not v|you3=have v|jiang3=speak v:resc|wan2=finish.
   1|2|NEG 2|3|CJCT 3|0|ROOT 4|3|SRL 5|3|PUNCT

10. **PRED** can be used to code clefts, as in this example:
    咪 姐姐 就 是 这样 讲 的 .
    co|lai2 n:relat|jie3&DIM=elder_sister adv|jiu4=just v:cop|shi4=is pro|zhe4yang4=so_such v|jiang3=speak nom|de .
    1|6|COM 2|6|SUBJ 3|4|JCT 4|6|JCT 5|6|SUBJ 6|0|ROOT 7|6|POSS 8|6|PUNCT

11. **TOP** is the basic relation used in Chinese for verbless sentences and sentences with resumptive pronouns. In the case of verbless sentences, the ROOT of the clause is the predicate adjective or noun and the initial topic attaches to that element with the TOP relation, as in this example:

    物价 纽约 最 贵 .
    n|wu4jia4=commodity n-prop|niu3yue1=New_York adv|zui4=the_most adj|gui4=expensive .
    1|4|TOP 2|4|SUBJ 3|4|JCT 4|0|INCROOT 5|4|PUNCT

    In the case of resumptive pronouns, the initial topic attaches to the resumptive pronoun with the TOP relation

12. **PTOP** is a postposed topic, as in this example:
    这 个 是 什么 这 个 .
    pro|zhe4=this cl|ge4 v:cop|shi4=is pro:wh|shen2me=what pro|zhe4=this cl|ge4 .
    1|2|DET 2|3|SUBJ 3|0|ROOT 4|3|PRED 5|6|DET 6|3|PTOP 7|3|PUNCT

13. **POSTO** is the relation between a noun and a following postposition. The postposition is the head and the nominal object of the preposition is the dependent. The postposition is linked to the verb as JCT. The relation is marked on the noun, as

in this example:

```
他 跑 到 教室 里面 去 .
pro|ta1=he v|pao3=run v:resc|dao4=arrive v|jiao1shi4=classroom
post|li3mian4=inside v:dirc|qu4=go .
1|2|SUBJ 2|0|ROOT 3|2|SRL 4|5|POSTO 5|2|JCT 6|2|SRL 7|2|PUNCT
```

14. **PREPO** is the relation between a preposition and a following noun. The preposition is the head and the object of the preposition is the dependent. The preposition is linked to the verb as JCT, as in this example:

```
他 是 从 北京 来 的 .
pro|ta1=he v:cop|shi4=is prep|cong2=from n:geo|bei3jing1=Beijing
v|lai2=come nom|de .
1|2|SUBJ 2|0|ROOT 3|2|JCT 4|3|PREPO 5|2|CMOD 6|2|LINK 7|2|PUNCT
```

PREPO and PREP can be combined, as in this example:

```
养 在 什么 里面 ?
v|yang3=raise prep|zai4=at pro:wh|shen2me=what post|li3mian4=insid?
1|0|ROOT 2|1|JCT 3|2|PREPO 4|2|POSTO 5|1|PUNCT
```

12.3 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 be also 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 "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
```

Root	ROOT	verbal ROOT; relation between verb and left wall: v, adj, subsidiary verb (tense bearing element) *taberu.* 'I'll eat it.' 1|0|ROOT
	COPROOT	COPula ROOT; copula with noun, adjectival noun, or sentence nominalizer (*no da*) *koko da.* 'it's here.' 1|2|PRED 2|0|COPROOT
	PREDROOT	nominal ROOT (without copula); includes adv, co, and quant, as well as verbal nouns and adjectival nouns in root position. *koko*. 'it's here.' 1|0|PREDROOT
	CPREDROOT	nominal ROOT with a sentence nominalizer in root position (ptl:snr|no=SNR) uma no chiisai no. 'a small horse' 1|2|MOD 2|3|CASP 3|4|CMOD 4|0|CPREDROOT
	CASPROOT	CASe Particle ROOT (the case particle is the head) dare ga? 'who?' 1|2|SUBJ 2|0|CASPROOT
	TOPPROOT	TOPic Particle ROOT *kore wa?* 'and what about this one?' 1|2|TOP 2|0|TOPPROOT
	FOCROOT	FOCus Particle ROOT *kore mo?* 'this one, too?' 1|2|FOC 2|0|FOCPROOT
Topic	TOP	TOPicalization, (for convenience the root of the sentence is considered to be the head) *kore wa yomenai.* 'I can't read it.' 1|2|TOP 2|3|TOPP 3|0|ROOT

	CTOP	finite Clausal TOPic (head of the clause is ptl:snr|no) *iku no wa ii kedo [...]* 'it's ok to go, but…' 1|2|CMOD 2|3|CTOP 3|4|TOPP 4|5|COMP 5|6|CPZR
	FOC	FOCus (followed by ptl:foc; *mo, shika, bakari, hodo* etc.) *kore mo yonda.* 'he read this one, too.' 1|2|FOC 2|3|FOCP 3|0|ROOT
Arguments	SUBJ	nonclausal SUBject Jon ga tabeta. 'John ate it.' 1|2|SUBJ 2|3|CASP 3|0|ROOT
	CSUBJ	finite Clausal SUBJect (head of the clause is ptl:snr) taberu no ga ii. 'it's good to eat it.' 1|2|CMOD 2|3|CSUBJ 3|4|CASP 3|0|ROOT
	OBJ	accusative OBJect *hon o yonda.* 'he has read the book.' 1|2|OBJ 2|3|CASP 3|0|ROOT
	COBJ	finite Clausal accusative OBJect *taberu no o yameta.* 'he stopped eating.' 1|2|CMOD 2|3|COBJ 3|4|CASP 4|0|ROOT
Adjuncts	JCT	adJunCT (Postpositional or adverbial phrase) *gakkoo kara kaetta.* 'he came back from school.' 1|2|JCT 2|3|POSTP 3|0|ROOT yukkuri shabetta. 'he talked slowly.' 1|2|JCT 2|0|ROOT
	CJCT	finite Clausal adJunCT *ochita no de taberu.* 'I'll eat with the one that had fallen down. 1|2|CMOD 2|3|CJCT 3|4|POSTP 4|0|ROOT
	XJCT	nonfinite clause as adJunCT (tabe-reba, -tara, -te, -cha, -tari; oishi-ku; shizuka ni) *kaeseba ii.* 'it's ok to give it back.' 1|2|XJCT 2|0|ROOT
Clause conjunction	CPZR	ComPlementiZeR (subordinating conjunctive particle; ptl:conj|) *osoi kara kaeru.* 'I'll go home because it's late.' 1|2|COMP 2|3|CPZR 3|0|ROOT
	ZCPZR	Zero-ComPlementiZeR (sentence introducing conjunction); head is always the root *dakara kaeru.* 'that's why I'll go home.' 1|2|ZCPZR 2|0|ROOT
	COMP	finite clausal verb COMPlement (before ptl:conj| and quot|to) *osoi kara kaeru.* 'I'll go home because it's late.' 1|2|COMP 2|3|CPZR 3|0|ROOT
	QUOTP	QUOTation Particle after nominal or verbal phrase *kaeru to iimashita.* 'he said he would go home.'

		13COMP 21QUOTP 30ROOT
	ZQUOT	Zero-QUOTative (sentence introducing quotative marker) *tte iu ka […]* 'in other words' 1|2|ZQUOT 2|3|COMP 3|4|CPZR
Nominal head	MOD	nonclausal MODifier (of a nominal) *Papa no kutsu ga atta.* 'there are Dad's shoes.' 1|2|MOD 2|3|CASP 3|4|SUBJ 4|5|CASP 5|0|ROOT
	CMOD	finite Clausal MODifier of a nominal; the dependent is a finite verb, adjective or adj noun with copula akai kuruma o mita. 'he saw a red car.' 1|2|CMOD 2|3|OBJ 3|4|CASP 4|0|ROOT
	XMOD	nonfinite clausal MODifier of a nominal (adn|) *kore to onaji mono ga […]* 'a thing similar to this one' 1|2|JCT 2|3|POSTP 3|4|XMOD 4|5|SUBJ 5|6|CASP
	COORD	COORDination, second noun is the head; (ptl:coo|) inu to neko o katte iru. 'he has a dog and a cat.' 1|2|COORD 2|3|COOP 3|4|OBJ 4|5|CASP 5|6|XJCT 6|0|ROOT
NP structure	PRED	nominal PREDicate before copula or QUOT *tabeta hito da.* 'he is the one who ate it.' 1|2|CMOD 2|3|PRED 3|0|COPROOT
	CPRED	finite Clausal PREDicate before copula (*no da*) *taberu no da.* 'in fact, he'll eat it.' 1|2|CMOD 2|3|CPRED 3|0|COPROOT
	CASP	CASe Particles (ptl:case; *ga, o*) *hon o yonda.* 'he read the book.' 1|2|OBJ 2|3|CASP 3|0|ROOT
	POSTP	POSTpositional Particles (ptl:post; *ni, de, kara, made, to*) *Papa ni ageta.* 'he gave it to Dad.' 1|2|JCT 2|3|POSTP 3|0|ROOT
	ATTP	ATTributive Particle Papa no kutsu (ga) 'Dad's shoes are…' 1|2|MOD 2|3|ATTP 3|4|SUBJ
	ATTP-SUBJ	ATTributive Particle in SUBJect position with head-noun elided *Papa no ga atta.* 'here is Dad's one.' 1|2|MOD 2|3|ATTP-SUBJ 3|4|CASP 4|0|ROOT
	ATTP-OBJ	ATTributive Particle in OBJect position *Papa no o mita.* 'I saw Dad's one.' 1|2|MOD 2|3|ATTP-OBJ 3|4|CASP 4|0|ROOT
	ATTP -JCT	ATTributive Particle in ADJunct position *Papa no de asonda.* 'he played with Dad's one.'

		1|2|MOD 2|3|ATTP-JCT 3|4|POSTP 4|0|ROOT
	ATTP -PRED	ATTributive Particle in predicate position *Papa <u>no</u> da.* 'it's Dad's one.' 1|2|MOD 2|3|ATTP-PRED 3|0|COPROOT
	ATTP-TOP	ATTributive Particle in TOPic position *Papa <u>no</u> wa agenai.* 'I won't give you Dad's one.' 1|2|MOD 2|3|ATTP-TOP 3|4|TOPP 4|0|ROOT
	ATTP-FOC	ATTributive Particle in FOCus position *Papa <u>no</u> mo agenai.* 'I also won't give you Dad's one.' 1|2|MOD 2|3|ATTP-FOC 3|4|TOPP 4|0|ROOT
	TOPP	TOPic Particle (ptl:top; *wa*) *kore <u>wa</u> yomenai.* 'I can't read this.' 1|2|TOP 2|3|TOPP 3|0|ROOT
	FOCP	FOCus Particle (ptl:foc; *mo, shika, bakari, hodo* etc.) *kore <u>mo</u> yonda.* 'I read this one, too.' 1|2|FOC 2|3|FOCP 3|0|ROOT
	COOP	COOrdination Particles (ptl:coo; *to, ya* etc.) *inu <u>to</u> neko ga […]* 'dogs and cats are…' 1|2|COORD 2|3|COOP 3|4|SUBJ 4|5|CASP
	QUANT	QUANTifier (incl. classifiers and counters) *banana <u>sambon</u> tabeta.* 'he ate three bananas.' 1|3|OBJ 2|3|QUANT 3|0|ROOT
	ENUM	ENUMeration, without coordinating particle *<u>ichi</u> <u>ni</u> sanko da.* 'there are 1, 2, 3 of them.' 1|2|ENUM 2|3|ENUM 3|4|PRED 4|0|ROOT
	NAME	string of proper NAMEs, second name is the head *<u>Kameda</u> Taishoo ga kita.* 'Taishoo Kameda arrived.' 1|2|NAME 2|3|SUBJ 3|4|CASP 4|0|ROOT
	DATE	string of DATEs, last element (day) is the head <u>rokugatsu</u> tsuitachi ni kita. 'he came on June 1st.' 1|2|DATE 2|3|JCT 3|4|POSTP 4|0|ROOT
Others	SMDR	sentence final Sentence MoDifieR (smod| ; *mitai, jan, rashii* etc); for convenience, the tense bearing verb is considered to be the head *kaetta <u>mitai</u>.* 'it seems he went home.' 1|0|ROOT 2|1|SMDR
	SFP	Sentence Final Particle (including the use after arguments and adjunct) *kuru <u>ne</u>.* 'he'll come, won't he?' 1|0|ROOT 2|1|SFP

	COM	COMmunicator; (co:i| co:g|) including isolated final particles, sentence modalizers and onomatopoeias; head is always set to 0 *anoo tabeta.* 'err..I ate it.' 1|0|COM 2|0|ROOT
	VOC	VOCative ; head is always set to 0 *Taishoo ‡ aka.* 'Taishoo, it's red.' 1|0|VOC 2|1|VOCP 3|0|PREDROOT
Punctuation	PUNCT	sentence boundary (sentence ends; .!? etc.); the root is the head *iku .* 'I'll go.' 1|0|ROOT 2|1|PUNCT
	RDP	Right Dislocation boundary (dloc|„=DISLOC)*;* dislocation follows; the root is the head *mita „ fuusen ?* 'the balloon, did you see it?' 1|0|ROOT 2|1|RDP 3|1|OBJ 4|1|PUNCT
	VOCP	VOCative marker (voc|‡=VOC); head is the preceding vocative *Taishoo ‡ mite !* 'Taishoo, look!' 1|0|VOC 2|1|VOCP 3|0|ROOT 4|3|PUNCT

Berlitz. (2005). *Berlitz Italian verbs handbook: 2nd Edition*. New York, NY: Berlitz.

Bloom, L. (1973). *One word at a time: The use of single word utterances*. The Hague: Mouton.

Bybee, J., & Hopper, P. (2001). *Frequency and the emergence of linguistic structure*. Amsterdam: John Benjamins.

Crain, S. (1991). Language acquisition in the absence of experience. *Behavioral and Brain Sciences, 14*, 597-611.

Edwards, J. (1992). Computer methods in child language research: four principles for the use of archived data. *Journal of Child Language, 19*, 435-458.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition, 48*, 71-99.

Hausser, R. (1999). *Foundations of computational linguistics: Man-machine communication in natural language*. Berlin: Springer.

Hyams, N., & Wexler, K. (1993). On the grammatical basis of null subjects in child language. *Linguistic Inquiry, 24*(3), 421-459.

MacWhinney, B. (1975). Pragmatic patterns in child syntax. *Stanford Papers And Reports on Child Language Development, 10*, 153-165.

MacWhinney, B., & Leinbach, J. (1991). Implementations are not conceptualizations: Revising the verb learning model. *Cognition, 29*, 121-157.

MacWhinney, B., Leinbach, J., Taraban, R., & McDonald, J. (1989). Language learning: Cues or rules? *Journal of Memory and Language, 28*, 255-277.

Meisel, J. (1986). Word order and case marking in early child language. Evidence from simultaneous acquisition of two first languages: French and German. *Linguistics,*

24, 123-185.

Mintz, T. H., Newport, E. L., & Bever, T. G. (2002). The distributional structure of grammatical categories in speech to young children. *Cognitive Science, 26*, 393-424.

Parisse, C., & Le Normand, M. T. (2000). Automatic disambiguation of the morphosyntax in spoken language corpora. *Behavior Research Methods, Instruments, and Computers, 32*, 468-481.

Plunkett, K., & Marchman, V. (1991). U-shaped learning and frequency effects in a multi-layered perceptron: Implications for child language acquisition. *Cognition, 38*, 43-102.

Quirk, R., Greenbaum, S., Leech, G., & Svartvik, J. (1985). *A comprehensive grammar of the English language*. London: Longman.

Radford, A. (1990). *Syntactic theory and the acquisition of English syntax*. Oxford: Basil Blackwell.

Rice, S. (1997). The analysis of ontogenetic trajectories: When a change in size or shape is not heterochrony. *Proceedings of the National Academy of Sciences, 94*, 907-912.

Rose, Y., MacWhinney, B., Byrne, R., Hedlund, G., Maddocks, K., O'Brien, P., & Wareham, T. (2005). Introducing Phon: A Software Solution for the Study of Phonological Acquisition. In D. Bamman, T. Magnitskaia, & C. Zaller (Eds.), *30th Annual Boston University Conference on Language Development* (pp. 489-500). Somerville, MA: Cascadilla Press.

Sagae, K., MacWhinney, B., & Lavie, A. (2004a). Adding syntactic annotations to transcripts of parent-child dialogs. In *LREC 2004* (pp. 1815-1818). Lisbon: LREC.

Sagae, K., MacWhinney, B., & Lavie, A. (2004b). Automatic parsing of parent-child interactions. *Behavior Research Methods, Instruments, and Computers, 36*, 113-126.

Siskind, J. M. (1999). Learning word-to-meaning mappings. In J. Muure & P. Broeder (Eds.), *Cognitive Models of Language Acquisition*. Cambridge, MA: MIT Press.

Stromswold, K. (1994). Using spontaneous production data to assess syntactic development. In D. McDaniel, C. McKee, & H. Cairns (Eds.), *Methods for assessing children's syntax*. Cambridge, MA: MIT Press.

Tomasello, M. (2003). *Constructing a first language: A usage-based theory of language acquisition*. Cambridge: Harvard University Press.

van Kampen, J. (1998). Left branch extraction as operator movement: Evidence from child Dutch. In S. Powers & C. Hamman (Eds.), *The acquisition of scrambling and cliticization*. Norwell: Kluwer.

Wexler, K. (1998). Very early parameter setting and the unique checking constraint: A new explanation of the optional infinitive stage. *Lingua, 106*, 23-79.