The University of Georgia
ARTIFICIAL INTELLIGENCE CENTER

CASPR Research Report 2005-01

DETERMINING SYNTACTIC COMPLEXITY USING VERY SHALLOW PARSING

Matthew J. Voss

caspr
COMPUTER ANALYSIS OF SPEECH FOR PSYCHOLOGICAL RESEARCH

Artificial Intelligence Center
The University of Georgia
Athens, Georgia 30602-7415 U.S.A.
www.ai.uga.edu/caspr

Determining Syntactic Complexity Using Very Shallow Parsing

by

Matthew J. Voss

(Under the direction of Dr. Michael A. Covington)

Abstract

This thesis describes a rule-based computer program, the Shallow Syntactic Complexity Analyzer (ShaC), for determining the syntactic complexity of English-language text. Syntactic complexity is determined by comparing strings of text to templates. The templates were constructed following the modified D-Level scale (Covington et al., 2004), which ranks syntactic complexity based on the age at which young children first acquire various syntactic structures. The later a structure is acquired, the higher ranking it gets. ShaC is unique in that it attempts to give a good estimate of syntactic complexity without doing a deep syntactic analysis. Such a detailed analysis would be time consuming; ShaC uses heuristics and generalizations to greatly simplify the task at hand. The result is a quick and efficient method for estimating syntactic complexity. ShaC score correlates highly with D-Level. Sentences with a verb taking a finite complement, verbs taking an *-ing* complement, verbs taking a non-finite complement, and comparatives drive the correlation.

Index words: Syntax, Syntactic Complexity, Sentence Complexity, Corpus Linguistics, Shallow Parsing, D-Level, Language acquisition, Language development

Determining Syntactic Complexity Using Very Shallow Parsing

by

Matthew J. Voss

B.A., The University of Georgia, 2005

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2005

Determining Syntactic Complexity Using Very Shallow Parsing

by

Matthew J. Voss

Dedication

This thesis is dedicated to my parents, and to all those who helped make it possible.

Acknowledgements

While writing this thesis, I was supported by a research contract with GlaxoSmithKline Research and Development Ltd., Michael A. Covington, principal invesigator.

I would like to thank Cati Brown and Congzhou He for helping with questions about hand ranking texts, Joe McFall and Eric Morris for helping to hand rank texts, Dr. Paula Schwanenflugel for reading a preliminary draft and saving the day at the last minute, Dr. Zachary Estes for critical statistics help, Salena Sampson for help, support, and valuable insights throughout, Dr. Donald Nute for getting me interested in Artificial Intelligence and supporting me throughout, and Dr. Michael Covington for support, time, caring, advice, and so much over the years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Sentence complexity has been used frequently as an indicator of language development and degradation. To date most complexity analysis has been done by hand. This thesis presents a rule-based computer program with heuristics, a Shallow Syntactic Complexity Analyzer (ShaC), for analyzing syntactic complexity of a text. The ShaC score is shown to correlate with D-Level scale (Rosenberg and Abbeduto, 1987; Cheung and Kemper, 1992; Covington et al., 2004). Experimental results show sentences with a verb taking a finite complement, verbs taking an *-ing* complement, verbs taking an infinitive complement, and comparatives are the most reliable indicators of complexity.

1.2 Sentence Complexity versus Syntactic Complexity

Because it will play an important role later, I make a distinction between sentence complexity and syntactic complexity. Sentence complexity is a measure applied only to complete sentences. Syntactic complexity on the other hand can be applied to sentences or larger chunks of text. It ignores sentence boundaries, and instead finds indicators of complexity wherever they occur in the text.

ShaC looks at syntactic complexity, not sentence complexity. This is somewhat unconventional but is motivated by the observation that sentence boundaries are often unclear or subjective. Biber et al. (2000) point out that different types of text use punctuation differently. Transcription uses specialized punctuation, and individual transcribers use different punctuation. In older texts the author may have conjoined more sentences with semi-colons than is habitual today. If what defines a sentence boundary varies across texts then it is difficult to do any analysis that parses (or requires) sentences for analysis.

1.3 Why is Syntactic Complexity Worth Measuring?

Syntactic complexity metrics find many uses in disciplines across the cognitive sciences: to measure readability, to assess cognitive impairment in schizophrenia (Thomas et al., 1996b) and Alzheimer's disease (Snowdon et al., 1996), and to track language development in young children (Rosenberg and Abbeduto, 1987; Cheung and Kemper, 1992; Covington et al., 2004; Lee, 1974).

Findings indicate that various measures of syntactic complexity are highly correlated with working memory as assessed through tests of forward and backward digit span (Barch and Berenbaum, 1994; Cheung and Kemper, 1992). Most notably, in a review of complexity metrics Cheung and Kemper (1992) have shown correlations between age, working memory, and syntactic complexity, suggesting that as one grows older, both working memory and syntactic complexity decrease. Thus syntactic complexity might suffice as an indirect method of detecting working memory problems. There is however a large body of evidence suggesting that working memory is for syntactic processing is largely separate from that for other processes (Thomas et al., 1996b).

ShaC, and the push towards quick efficient syntactic analysis of text, is motivated by some findings regarding the use of complexity metrics in predicting mental illness. For example, simple syntax early in life may be a strong indicator of the potential to develop Alzheimer's disease later in life (Snowdon et al., 1996; Kemper et al., 2001). Similar findings by King et al. (1990) and Morice and Ingram (1983) show a trend in schizophrenia of degrading syntactic complexity unseen in normal individuals. They suggest that like in Alzheimer's disease, delayed acquisition or reduced syntactic complexity early on may be important indicators for children at risk for schizophrenia. These findings, supported by a large body of other work on syntax and schizophrenia (Thomas et al., 1996b; Barch and Berenbaum, 1997; Morice, 1985) indicate that by analyzing the syntax of transcripts of speech one can glean an extraordinary amount of information about a speaker. It is thus imperative to determine what techniques are best for analyzing syntactic complexity and to develop quick methods of doing the assessment. A proper automated tool would save researchers extraordinary amounts of time and money. This thesis is a step in that direction.

1.4 How Do You Measure Syntactic Complexity?

Different metrics are sensitive to different aspects of complexity, from parts of speech (Scarborough, 1990) to clausal embedding (Frazier, 1985; Yngve, 1960). One must choose the right metric for their purposes.

Our purpose here is to do analysis as a means of assessing cognitive impairment. Many different metrics have been used for this purpose including the following: Yngve Depth and Maximal Yngve Depth (Yngve, 1960), which measure the depth of left branching, generally from the subject, in sentences; Brief Syntactic Analysis (Thomas et al., 1996a), which makes multiple passes over a sentence to count sentence length, pausing, mean length of utterance, mean clauses per utterance, and mean maximum depth of embedding; Mean Clauses per Utterance (MCU) (Kemper et al., 1989), which simply counts the number of clauses in each utterance; and Frazier Count, another measure of clausal embedding[1] (Frazier, 1985). One way to detect the relevant features is to do a full parse of each sentence then extract the relevant complexity indicators from the parsed text.

This approach has some disadvantages. One problem is that rival linguistic theories will provide different analyses of given sentences. Another is that humans make mistakes. In parsing large amounts of text only a trained linguist will provide the accuracy required. Such a linguist may not always be available, and even linguists make mistakes.

[1]See Cheung and Kemper (1992) for a good review of these and other metrics.

The D-Level scale provides an alternative to these analyses which avoids some of their pitfalls but introduces some of its own.

1.5 What Is D-Level?

The D-Level scale is a scale for measuring sentence complexity, developed by Rosenberg and Abbeduto (1987) and revised by Cheung and Kemper (1992) and Covington et al. (2004). The scale is based on the age at which children acquire certain syntactic structures in their speech. The later children acquire a certain structure, the higher it appears in the scale. The revised scale is shown in Appendix A.

1.6 Why D-Level?

Cheung and Kemper (1992) suggest that the D-Level scale is, like most other methods, difficult to use when rating sentences. A rater must look over each sentence very carefully for each structure specified by the scale. D-Level is, however, attractive because it has a proven track record looking at mental impairment. It came about as a reflection of a child's linguistic development but has since been used to detect early symptoms of Alzheimer's disease, as well as to detect general syntactic simplification in older adults (Snowdon et al., 1996; Kemper et al., 2001). As Covington et al. (2004) point out, the importance of this result should not be underestimated, and a review of the metrics used is very much in order. Furthermore, in a test of a variety of complexity metrics, Cheung and Kemper (1992) found D-Level to be one of the most reliable metrics tested. They suggest its strength lies in its ability to distinguish between types of clausal embedding, giving the analyzer a more fine-grained look at the syntactic makeup of a text or transcript.

Another strength of the D-Level scale reveals itself upon analyzing a few sentences. Contrary to Cheung and Kemper's (1992) suggestion, one does not need to go over the sentence carefully in every case. Some of the structures the D-Level scale identifies seem to "pop out" of the text. They follow a pattern. We can simply look for these patterns to detect

D-Level. In looking for these patterns one can either look for words or phrases as keys to the level of the sentence. This idea is one that lends itself to a computational implementation, as shown in the next chapter.

Chapter 2

Program

A computer implementation of a complexity scale presents several advantages. Firstly, it will be fast. Second, it will be reliable; it will always give the same analysis to the same sentence. A human rater is prone to error. Third, a computer will not be biased to expect that one text will be simpler than another. The program is intended to be used to distinguish healthy from mentally ill subjects. Knowledge that a transcript is from one or the other may bias ratings. The following motivates and explicates ShaC.

2.1 A Parsing Approach to Complexity Analysis

Before presenting ShaC, some alternatives are considered. In the last chapter, parsing each sentence and doing the analysis from parsed text was presented as a viable option. However, a computer implementation of this approach has several problems. The foremost problem is that parsing is a difficult and unsolved problem. Even highly accurate parsers make mistakes. This means for quite a bit extra work, the system is still prone to error.

Another problem is that parsing is time consuming and computationally expensive. Then even once the parser generates a tree, D-Level analyses don't need the complete tree. They look only for specific things in each sentence. Ideally, only the necessary structures would be parsed.

2.2 Other Computational Approaches to Syntactic Analysis

There have been a few larger scale attempts to use computers to extract sentence and syntactic complexity information from texts. While the specific features of language measured by these programs are different than ShaC, some of the techniques they use are similar to ones used by ShaC. The following provides a review of these programs.

Channell (2003) presents a computer implementation of Developmental sentence scoring (DSS) (Lee, 1974), called Computerized Profiling (CP). DSS is used to evaluate a child's use of standard English. It focuses on grammatical complexity rather than syntactic complexity. DSS tallies occurrences of words in 8 grammatical categories, and assigns points based on the complexity of the category. It awards an additional point for sentences that are syntactically and semantically adult-like. Adult-like sentences consist of those that are complete, with at least a subject and verb. CP and some related programs, for example CLAN (MacWhitney, 2002), do morphological analysis and part-of-speech tagging to determine the grammatical categories of all the words in a text. CP uses a multi-stage process that combines the efforts of several different programs, and requires that input files be transcribed and formatted in a particular way. It skips sentences that contain unidentifiable words and requires that a human analyzer follow up the computer analysis to correct errors and catch things the program missed. The intent of the program is simply to reduce the amount of work that needs to be done by hand. CP and DSS recognize that a good indication of the complexity of a speech sample can be determined by looking simply for words of particular grammatical categories, which are relatively easy to detect. ShaC follows a similar assumption.

2.3 Shallow Parsing for Complexity Analysis

The relevant structures of the D-Level scale can be detected by looking for key phrases and words with chunk parsing and shallow parsing. Chunk parsing (Abney, 1991) was originally introduced as a parsing method that corresponded more closely with how humans parse text. "Chunks" correspond roughly to prosodic units, or pauses, in speech. These units correspond roughly to noun phrases, verb phrases, adjective phrases, and so on. Shallow parsing is detection of indicators of phrase structure without necessarily constructing that full structure. Words often are such indicators.

Parsing in this manner means that there is no overall sentence structure, just phrases. Furthermore, rules for each phrase type are non-recursive. Simple phrases are all the parser looks for.

2.4 Why Shallow and not Deep Parsing?

As noted above, the D-Level scale considers a wide variety of syntactic structures important to the analysis of complexity. Many of the structures that the D-Level scale detects are patterns that are identifiable by looking only at a small portion of the sentence. Since the majority of structures it looks for can be detected without doing a full parse, it is more efficient to do a shallow parse and get an approximation of the complexity of a text. Those structures that need a deep parse to be detected are fairly rare. Many of the sentence types analyzable with D-Level are not commonly heard in everyday speech. Rosenberg and Abbeduto (1987) estimated only thirty percent of all the sentences they analyzed were "complex" sentences (i.e., higher than level 0 on the scale).

This thesis outlines a program, ShaC, that will give a good approximation of the general linguistic performance of a speaker. ShaC is intended to analyze bodies of text and paint a general picture of the syntactic complexity of the text. This means that it will not detect everything the D-Level scale detects. The purpose of ShaC is to automate some syntactic analysis that can be done quickly with few resources: to glean as much syntactic information as possible without parsing. It is motivated by the intuition that while human language has a high potential for variation, speakers tend to use the same words, phrases, and syntactic structures repeatedly. From this assumption, I hypothesize that one can approximate the analysis done with D-Level scale by looking for patterns of words and phrases in text transcripts. The question addressed by ShaC will be how much syntactic complexity can be estimated by a shallow look at the text. D-Level is chosen as a model on which to base this program because one can analyze many of the structures D-Level does in a simple and efficient manner. ShaC is not intended to be a stand-alone tool for analysis of speech, but meant to give a look at some of those features that comprise the D-Level scale. It does this by adopting some simple heuristics which take the form of templates indicating strings of word categories that are indicators of complexity.

2.5 A Note on Sentences

If ShaC is to disregard sentence boundaries, as discussed in Section 1.2 notions such as "relative clause in object position" and "relative clause in subject position", which are important when looking at clausal embedding for example, become difficult or meaningless to determine.

The following utterance illustrates the problem:

(2.1) *I don't know the boy that you saw. The boy that I know has red hair.*

It is a plausible utterance. Here the period easily distinguishes the two sentences. The first has a relative clause in object position and is ranked a level 3 sentence while the second has a relative clause in subject position and is ranked a level 6 sentence. One might have transcribed it differently:

(2.2) *I don't know the boy that you saw; the boy that I know has red hair.*

We might then count semi-colons and periods as indicators of sentence breaks, but this limits the applicability of ShaC. In some transcription punctuation is used to indicate pauses, breaks, and other utterances. Since we want to establish a broad framework on which one might build, we ignore sentence boundaries.

The following illustrates another problem with sentence boundaries:

(2.3) *But I have a terrible secret: I only managed to push the right buttons by accident.*

This is one sentence, two clauses conjoined by a colon. It easily could have been two separate sentences.

A common division used to solve this problem is the T-unit (Hunt, 1965), defined as the "shortest grammatically allowable sentences into which writing can be segmented or minimally terminable unit." Essentially a T-unit is a clause and all of its subordinating clauses. To detect a clause and its subordinating clauses, one must parse the whole sentence. As the approach proposed here eschews parsing, T-units are not a useful way to segment a text.

2.6 Modifying the D-Level Scale

The full D-Level scale as outlined in Appendix A cannot be easily detected by template matching. A primary issue (discussed in Section 2.5) is how to determine sentence boundaries. If the task at hand is to look for syntactic complexity, not sentence complexity, the D-Level scale needs to be changed in several respects.

Level zero sentences are assumed basic. As will be seen later, points are assigned based on level. These would receive zero points and thus not affect the syntactic complexity score in any way. Furthermore if sentence boundaries are ignored then it becomes difficult to distinguish simple sentences from simple phrases. Since level 0 structures, indicating simple sentences, are ignored, the sentence *the cat ate the mouse that the farmer trapped* is ranked level 3 because it only includes a level 3 structure, a relative clause in object position.

Level two sentences include all conjunctions. People tend to use conjunctions to conjoin various phrases as well as at sentence boundaries as a way of guiding the listener. Since ShaC ignores sentence boundaries, this distinction between conjunctions used between phrases and those used at the beginning of sentences is lost, and so level 2 sentences are not included in the list of templates used by ShaC.

Level three includes the following: relative clause modifying the object of the main verb (*The cat ate the mouse that the farmer trapped*), nominalization in object position (*I appreciate his changing the lightbulb*), and finite clause as the object of the main verb (*He said I am not good enough*). Unfortunately levels 3 and 6 are not easily distinguishable by template matching. Level 6 sentences include: relative clause modifying the subject of the main verb (*The man that the boy saw was wearing a black coat*), embedded clause serving as the subject of the main verb (*That he came to the party surprised me*), and nominalization serving as the subject of the main verb (*His crying is getting annoying*). Using shallow parsing and no sentential boundaries, the distinction between object and subject positions cannot easily be determined. Take the following sentences:

(2.4) *I know what the girl likes. The boy that she saw said she liked diamonds.*

(2.5) *The girl likes the boy that she saw.*

In the first *the boy that she saw* marks a relative clause in subject position and so would be ranked a level 6 structure. In the second *the boy that she saw* marks a relative clause in object position and so would be ranked a level 3 structure. The limitations of the current approach become apparent. Since the same pattern in one situation is ranked a level 6 and in another is ranked a level 3, and since with shallow parsing there is not a good way to distinguish the two, we must make a sacrifice. Thus all level 6 sentences are ranked as level 3 sentences, since these are easy to detect using the following template:

```
noun,verb,noun,complementizer
```

In looking at several sample texts I found no examples of level 6 sentences. This suggests they are rare enough to be excluded without losing too much of the effectiveness of the analysis. It should be remembered that even level 6 sentences are getting some ranking: ShaC notes there is some complexity indicator at these points, but does not rate it as complex as the original D-Level scale.

Level 4 includes non-finite complements with their own understood (or explicit) subjects. In particular sentences like *I consider John a friend* are hard to include in a template. Templates for sentences such as these were originally included in the analysis but have been removed because ShaC misanalyzed them in most cases. This should not present a problem, however, since constructions of that type are rare especially in spoken English.

Level 7 reflected combining two or more of levels 1 through 6 in a single sentence. Since the analysis is of syntactic complexity, level 7 is no longer needed. Of note, level 7 is in large part the reason for adopting the syntactic analysis approach. As Cheung and Kemper (1992) point out, the D-Level scale, amongst other metrics, is sensitive to sentence length. The longer a sentence, the more complex it is. Since English speakers tend to join sentences with 'and', most sentences on a strict D-Level analysis will be ranked level 7. For this reason coordinating conjunctions are not considered as complexity indicators.

2.7 Program Overview

ShaC is a rule-based pattern matcher. Given a template, it will find all strings in a text that match the template. It is supplied with templates corresponding to structures in the D-Level scale. The D-Level scale has several different sentence types that it ranks at each level. At level 3 for example, there are 9 different sentence types it is looking for. There is one template for each of these sentence types. Each template specifies the nodes in a Finite State Automaton. A series of rules specifies the conditions under which ShaC should move from one node to the next.

The choice to do a rule-based system is motivated by the idea that people use English in regular ways; strange or deviant speech and writing is rare. Rules will do well in picking out some regularities, and general trends in language.

ShaC is unique in that it uses a minimal amount of information to do its analysis. It does not rely on punctuation, it does not do part of speech tagging, and it does not do a deep syntactic analysis. The only parsing it does is of some chunks. It was designed to handle large amounts of data in a short amount of time, to be simple and robust.

2.8 Program Components

ShaC was written in Prolog for a number of reasons. First, ShaC is rule-based. Prolog is an ideal language for dealing with such systems. Secondly, there are a wide variety of natural language processing tools freely available for Prolog, providing the infrastructure to construct the system without having to custom build tools.

That being said, the tools that were used were not specifically designed for the present task and needed heavy modification. The following presents the tools used and addresses the modifications necessary to use them.

2.8.1 WordNet

WordNet (University, 2005) is a massive lexical database with a Prolog interface. It provides the lexicon and relevant part of speech information for each word to ShaC. This is perhaps a seemingly odd choice. There are plenty of decent part-of-speech taggers that would give part-of-speech information. The problem is that most of the tags they give are not useful for the present task. As will be shown later, the tags needed will have information reflecting part of speech as well as morphological and semantic properties of words. WordNet provides a relatively unadorned lexicon that can be manipulated to glean the relevant information.

The database has a few major drawbacks to overcome. First, there are too many entries. People tend to stick to common uses of common words especially in natural speech. Witzig (2003) provides a suite of tools for manipulating the WordNet database, including one that will extract a subset of WordNet based on word frequency. Each entry in the WordNet database, each sense of each word, includes information about its frequency of occurrence in a corpus. This tool will extract the x most frequent entries. Using this tool the database was reduced to approximately 30,000 entries.[1]

[1]Reducing the size of the lexicon and removing duplicate entries compacts the lexicon considerably; however, the approach taken to reduce the lexicon is in some sense not selective. In the present case, the lexicon was simply reduced to a lexicon of 30,000 words. This was done without paying much attention to the sample from which the frequency statistics were collected. If the corpus that was analyzed to get these statistics is comprised of texts from specific registers, then the reduced lexicon will reflect that bias.

A glance through the lexicon shows that in some sense it may be biased. Included in the 30,000 most common words in the corpus analyzed are entries for ‘no’ and ‘no.’ WordNet gives the same definitions for both of these indicating that the period in the second indicates a notational variant. Furthermore the example sentence for the most common sense is *his no was loud and clear*. Other examples might be *The answer is no*. In both of these cases the use of ‘no’ presumably refers to what someone said. The lexical entries do not reflect this fact. I suggest in each of the above ‘no’ was used in its adverbial sense and then referred to in conversation. WordNet attempts to get all the appropriate senses of all the words in its database. As can be seen here, the entries in WordNet

Second, the entries are not stored in the most useful form. ShaC interfaces with other tools that use `word/2` clauses to represent the lexicon, for example:

```
word(n,bacon).
```

The entries in the WordNet database are stored as `s/6` clauses where the part of speech is represented as the fourth argument and the lexical entry is represented as the third argument, for example:

```
s(_,_,bacon,n,_,_).
```

For each `s/6` clause a corresponding `word/2` clause was created to speed lexical retrieval. Furthermore, each word was reduced to one entry in the working lexicon. All alternative senses were removed. Only one entry for each word was needed.

Entries in WordNet can have multiple words, atoms connected by the '_',and capital letters. For use with ShaC, these atoms are converted to lists of atoms in lower case. So the following:

```
s(100048485,1,'French_leave',n,1,0).
```

becomes

```
word(n,[french,leave]).
```

2.8.2 The Closed-Class Lexicon

WordNet contains none of the closed-class words. A closed-class lexicon in Prolog (De Juan, 2004) was used to supplement the WordNet lexicon. Again the lexicon needed to be altered. Entries in the closed-class lexicon were as follows:

are not perfect, and are open to debate. Even with a reduced lexicon, one encounters errors such as these.

```
ccl('as long as', conjunction, [subordinating]).
ccl(of, preposition, []).
```

Each predicate gives the lexical entry, its category, and any subcategory. As with the `s/6` entries in WordNet, multiple word entries were represented as single atoms. To facilitate interaction with ShaC these were converted as follows:

```
ccl([as, long, as], conjunction, [subordinating]).
ccl([of], preposition, []).
```

2.8.3 ProNTo_Morph

ProNTo_Morph (Schlachter, 2003) is a morphological analysis tool for Prolog. I used this program essentially as is, adding exceptional morphological analyses where needed.

2.8.4 A Simple Chunk Parser

SCP (Brooks, 2003) is a simple chunk parser that follows Abney's (1991) algorithm. I used the algorithm essentially as is; however, modified the output to include a list of the words just parsed. I only parsed prepositional phrases.

2.8.5 ET: The Efficient Tokenizer

I used ET (Covington, 2003) to give a list of all of the words in a text file. ET strips out all punctuation and numbers. A hypothesis to be tested is that a reasonable approximation of the D-Level scale can be achieved without looking at all at punctuation.

2.9 Useful Heuristics

People use language in regular ways. One can approximate D-Level by noting these regularities and looking for them in text transcripts. The following sections provide an overview of the major heuristics used to detect features of the D-Level sentence types.

$$\begin{aligned} PP \rightarrow &\ P\ (P)\ DP \\ DP \rightarrow &\ (Det)\ (Det)\ (Adj)\ (Adj)\ NP\ (s) \\ DP \rightarrow &\ Pron\ (Adj)\ (Adj)\ NP \end{aligned}$$

Table 2.1: Prepositional Phrase Rules

2.9.1 Adjectives and Adverbs

The D-Level scale does not take adjectives or adverbs into account. This means essentially that they are "filler." Furthermore, adjectival and adverbial phrases for the most part consist entirely of adjectives and adverbs respectively. Instead of parsing adjectival and adverbial phrases, any adjectives and adverbs can be skipped assuming that the next word is likely one of the type ShaC is looking for.

2.9.2 Prepositional Phrases

Like adjectives and adverbs, prepositional phrases are largely used as "filler," providing extra information that does not change the syntactic complexity of a sentence according to the D-Level scale. This means that, with a few exceptions, prepositional phrases can be parsed and skipped. Prepositional phrases are kept simple in the spirit of shallow parsing. The rewrite rule for prepositional phrases is shown in Table 2.1.[2]

The rules for prepositional phrases exclude some common types of prepositional phrases. For example:

(2.6) *We went to the store by taking the train.*

[2]Note the use of DPs in these rules, and also that nothing rides on the adoption or abandonment of the DP-hypothesis (Abney, 1987). Adopting this convention is merely for convenience and does not affect the performance of ShaC. The above rules follow a simple grammar laid out by Abney (1991).

Phrases like *by taking* are excluded because they typically signify the presence of an adjunct of the main verb.

2.9.3 Non-finite Complements

Only certain verbs take non-finite complements. Levels 1, 4, and 5 of the D-Level scale look for non-finite complements. One option is to look for patterns of words following any verb. A template for level 1 might be:

```
verb, ving
```

where `ving` is a verb with an *-ing* ending. However this template will produce false positives. For example it would accept the following:

(2.7) *The boy fell trying to ride his bike.*

The key is that only certain verbs take a non-finite complement. Looking only for those verbs greatly increases the accuracy of ShaC. Appendix B includes a list of all the verbs used that take a non-finite complement. This is by no means a complete list; however, it does seem to cover the words encountered in sample texts.

2.9.4 Noun Phrases and Verb Phrases

The determiner phrases of Table 2.1 provide a good definition of the noun phrase as a chunk. Such phrases include determiners, nouns, modifiers, and perhaps the possessive clitic. Adjectives, as noted above, are skipped. The only other elements that might be in a noun phrase are determiners, nouns, and the possessive clitic. Any of these elements that ShaC encounters it can skip.

This saves a lot of work. Take for example a case where the template specifies a plural noun. One might parse the full noun phrase, then traverse the tree to the noun and check to see if the noun is plural. If it is not, a lot of time was spent parsing, only to fail.

Instead, if a noun is the target and the input string shows something that is a “child of” a noun phrase, then ShaC can advance to the next word, so long as the next word is also part of a noun phrase or another nonessential element. Suppose we are looking for a plural noun in the following:

(2.8) *The very big, bushy cats jumped.*

‘The’ is a determiner. The word ‘very’ is an adverb and ‘big’, and ‘bushy’ both adjectives, so they are passed over. The next word is ‘cats’. ShaC checks that it is plural, and succeeds. No parsing was done. In fact, ShaC saves a lot of work by not parsing.

There is one large disadvantage to this approach, which is tied to the choice to regard punctuation as unreliable. One of the example sentences for a verb with a non-finite complement with its own understood subject is:

(2.9) *I consider John a friend.*

A potential template for this type of sentence is:

```
vnfc, n, n
```

However, this template will produce numerous false positives, especially where the two noun phrases would be in different sentences , such as:

(2.10) *John hated the boy. He...*

This sentence is read into ShaC as a list of lowercase Prolog atoms:

```
[john,hated,the,boy,he]
```

This sentence matches the proposed template. Parsing complete noun phrases would allow a correct analysis. On the other hand, this type of sentence seems to be fairly rare in both text and speech. Of a large sample of sentences analyzed by ShaC, none actually fit this template. It was better to leave the template out.

The case is less complicated for verb phrases. Verb phrases typically consist of auxiliary verbs, main verbs, and modifiers. Adverbs are skipped. Verbs are similar. If we are looking for a verb, and see an auxiliary verb followed by an adverb, another auxiliary, or another verb, then the current verb is skipped.

2.9.5 Nominalizations

A good number of common nominalized verbs can be detected by looking at the ending of a given word. If the ending is either *-ing* or *-ion* and the stem is a verb, then the word is a nominalization. There are of course many exceptions to this rule: *betrayal*, *refusal*, *denial*, for example, all end in *-al*, and a rule could be created to detect nominalizations of this type. Other suffixes indicating nominalization include *-ment*, *-ity*, *-ness* (Biber et al., 2000).

More exceptions are words such as *notion* and *action* that are not nominalizations. These represent the limits of the proposed approach. But again these are exceptions to the rule. As such, even if ShaC counts them, they make up a small portion of nominalizations, which make up a small portion of words used in everyday texts. They should not affect ShaC's output considerably.

2.9.6 Detection of Adjuncts

A limitation of the current approach is that adjuncts are often difficult to distinguish from complements. Whether a given phrase is an adjunct or a complement is often largely a function of the verb. For example in

(2.11) *The boy gave the ball to the girl*

to the girl is a complement of the verb. In

(2.12) *The boy kicked the ball to the girl.*

to the girl is an adjunct. There are however a few indicators of an adjunct that we can use to increase the reliability of the analysis. ShaC looks for prepositions followed by *-ing* forms of verbs, for example.

Level five looks for nominalizations in adjunct not complement position. This particular structure can be found by looking for prepositions followed by nominalized verbs. This and other templates used are discussed later.

2.9.7 Verbs in plain, *-s*, *-ed*, and *-ing* form

For certain templates it is useful to make a distinction between verbs with different morphology. Three categories are useful. The first category is verbs in plain form. These match WordNet entries and so can be looked up directly with a `word/2` clause.

The second type of verb is any in in plain, *-s*, or *-ed* form. We find these by doing a morphological analysis. ProNTo_morph (Schlachter, 2003) returns three relevant tags. For verbs in *-s* and *-ed* form it returns '-s' and '-ed' tags. For verbs in third person singular form, which may be irregular, it returns '-sg3'. Similarly for *-ing* verbs, ProNTo_morph returns an '-ing' ending. This makes categorization straight forward.

2.9.8 *that*

The closed-class lexicon was meant to be nearly complete. As such, *that* is a subordinating conjunction. Without taking verb argument structure into account, it is often difficult to determine whether *that* is functioning as a subordinating clause, and thus should be ranked as level 5, or if it is introducing a finite or non-finite clause as the object of the main verb. That is determining if a sentence is a level 3 sentence or if it is a level 5 is difficult. Take the following sentences for example:

(2.13) *John knew that Mary was angry.*

(2.14) *John prayed that Mary was alright.*

In Example 2.13, *that Mary was angry* is the complement of *know*. In Example 2.14, *that Mary was alright* is arguably an independent sentence and *that* connects it to *John prayed*. Since this is the case, *that* is not considered a subordinating conjunction for D-Level analysis.

2.10 Templates

This section discusses the foundation of the approach to D-Level analysis taken here. Templates specify a sequence of elements to look for in a string of text. Only major elements are represented in templates. These include nouns, verbs, complementizers, subordinating conjunctions, and, in general, the heads of phrases. Appendix A summarizes the templates for each level in the right-hand column. There are templates for seventeen of the sentence types ranked by the D-Level scale. Also included in Appendix A is a table indicating what each of the symbols that make up each template mean. The hypothesis is that analysis based on these templates will give a reasonable estimation of the D-Level complexity of a text.

Reading the templates should be straight forward. ShaC accepts words from the input until it finds one matching the first symbol in the template. It then proceeds to look for the next symbol in the template, skipping those words it can, as discussed above. If ShaC finds a word that does not match the current symbol in the template, the remainder of the input stream is returned and the program fails. If ShaC reaches the end of the template, has found a string that matches the template, then it returns this string of words. An example will make this clearer.

Suppose the following is the input sentence:

(2.15) *The boy who likes to play slept all day.*

The task at hand is to compare each of our templates to this sentence. A level 1 template is:

```
vpsed, to, vpl
```

ShaC skips *the* and *boy* because they are not verbs in plain *-s* or *-ed* form. *like* is a verb of this type. There are no verbs of this type following it, so ShaC accepts a verb. The next word in the input, *to*, matches the next template, also *to*. ShaC accepts *to*. The next word in the input *play* matches the template `vpl`, and so ShaC accepts *play*. This completes the template. ShaC returns a level 1 ranking.

Another template, at level 3, is:

```
n, comp, vpsed
```

First ShaC is looking for a noun. The program skips *the* because it is a determiner and the next word is a noun. *Boy* is a noun. No other nouns follow. ShaC accepts *boy*. The next word in the input is *who*. *Who* is a comp. ShaC accepts *who*. The next word is *likes*. *Likes* is a `vpsed` verb so ShaC accepts it and succeed in matching the template.

2.11 Output

ShaC takes a text as input. For each template, it counts each string in the input that matches the template. This total number of strings matching a template is weighted by D-Level. The number of strings matching a level 3 template for example is multiplied by 3. All the weighted string counts are summed and the total is divided by the number of words in a text. The result is a complexity score for the text.

For example, in Example 2.15 ShaC found a level 1 construction, *likes to play*, and a level 3 construction *boy who likes*. There are nine total words in the text so a complexity score is computed as follows:

$$\frac{(1+3)}{9} = 0.44$$

The total complexity score for the text is 0.44. The formula more generally is:

(2.16) $$\frac{(L_1C_1 + L_2C_2 + L_3C_3 + \ldots + L_kC_k)}{W_t}$$

where L_k is the level of the template, C_k is the count of a template, and W_t is the number of words in the text. Note that this ranking method is one suggested way of getting ShaC scores. Tests in the following chapter will use a different method to compare D-Level to ShaC score.

Chapter 3

Experiment

3.1 Introduction

This chapter presents an experiment intended to validate the approach to analyzing D-Level outlined above. The main hypothesis to be tested is that one can approximate a D-Level analysis by using shallow parsing, no part of speech tagging, impoverished lexical knowledge, without statistical analysis, and using a rule-based system.

The cards are stacked strongly against this hypothesis. Section 2.9 above outlined many useful heuristics. Even a cursory glance at these heuristics brings to mind dozens of exceptions. Furthermore only twelve of the sentence types covered by the D-Level scale are ranked. There are no doubt many other sentence types that the D-Level scale can rank which are not represented by the example sentences.

Some evidence supports the idea that language follows particular trends. Rosenberg and Abbeduto (1987) point out in their original paper that the D-Level scale is not meant to correlate with frequency of use. Those structures higher in the scale are not necessarily used more frequently. The use of a structure probably correlates to its utility in expressing certain ideas. The authors found that only 30% of the sentences they analyzed were "complex", ranked higher than level 0.

The example of nominalization gives some indication of how much ShaC's performance will be affected by cutting corners. In a sample of 6.2 million words from the Longman-Lund Corpus covering texts from academic writing, fiction and speech, 2.5% of the words were nominalizations (Biber et al., 2000). By far the majority of those, 118,800 in 2.5 million words, were from academic writing. From speech samples there were 5650 nominalizations in 500,000 words. They are not used often. Detecting any at all is an accomplishment.

3.2 Methods

850 sentences from 45 texts including newspapers, magazines, fiction, academic texts, and blogs were hand ranked according to the D-Level scale in Covington et al. (2004) and analyzed by ShaC. The computer analysis treated sentences as individual texts to be analyzed and required that a human analyzer decide where to make sentential boundaries. For all texts periods were used as markers of sentential boundaries. Three raters[1] contributed to ranking all of the texts. An overview of the scores from ranking the texts by hand is given in Appendix C.

D-Level is a single number score 0 through 7 for each sentence. ShaC does not pay attention to sentential boundaries, produces a score for each text, and identifies individual indicators of complexity. The measure of interest is whether D-Level correlates in any way with the individual structures that the program finds. This correlation is useful over and above the correlation between D-Level and the single digit score the program gives because by looking at the individual complexity indicators, one can answer more questions about what it is in texts that D-Level is picking out and how well ShaC finds those same things. Again, one might look at the correlation between D-Level and ShaC score, but this relation only tells that D-Level scores can be mapped onto ShaC scores, not what is driving that relationship.

The correlation between D-Level and ShaC's output was determined as follows. ShaC looked at individual sentences in each text. The program counted the number of structures in each sentence corresponding to each of the templates in Appendix A along with sentence length. For each text, an average count for each template was determined. These averages comprise 45 data lines used to create and test a model. Correlations between D-Level and the

[1]Thanks to Eric Morris and Joe McFall for their help in ranking texts

seventeen template counts for each text from a portion of the data were assessed. A multiple linear regression was performed to find an equation that fit the data with D-Level as the solution to a multi-variate equation and significant templates as values in the equation. To illustrate, take the following example equation wherein d is D-Level:

$$d = xt_1 + yt_2 + zt_3 \tag{3.1}$$

If d is the average D-Level of a text and t_1, t_2 and t_3 are average template scores, then the job of the analysis is to find x, y, and z such that for any D-Level and any template scores, the resulting equation will hold to some small margin of error. The task was to determine the coefficients that produced an equation that best fit template values to D-Level.

Three different models were created using the data. In each case the data was divided in a different way to control for any homogeneity in the data sets. Originally the texts were ordered by register. The hypothesis is that different registers contain sentences of different types. Intuitively newspaper articles and fiction will use more verbs taking a finite complement, for example newspapers often quote directly what people have said:

(3.2) *"We have to take this ruling and try to figure out what it means to (the) program," said Robert Miller, a state health department spokesman.*

In this case *said* is counted as a verb taking a finite complement. Academic texts will contain considerably less of this sort of sentence. Thus the presence of this type of verb will not be a good indicator of complexity in those types of texts. It is to control for this effect that the different experiments were conducted.

The remaining portion of the data was then analyzed using these same significant factors, with the intent of answering two questions: which templates correlate significantly with D-Level and how well do they correlate? The hypothesis is that only certain templates will indicate D-Level, that in normal speech, we use certain of the structures D-Level looks for more frequently than others. Those we use most frequently will be a good estimate of the

Template	$p <$	r
-ing Complements	0.0478	0.3642
Finite Complements	0.0216	0.41793

Table 3.1: Model 1: Templates Correlating Significantly with D-Level

overall D-Level of a text. Thus not only can we estimate D-Level using general templates, just a few of theses will be enough to give a good estimate.

3.3 Model 1

The data used in this experiment were collected in two groups. Fifteen of the data points came from a wide variety of sources including children's fiction, blogs, fiction, newspapers, and academic texts. Thirty of the data points came from three registers: newspapers, adult fiction, and academic texts. The data was divided along these lines for purposes of making a model and testing it. The advantage to dividing the data this way is there is a smaller chance that the two groups are homogenous, thus if the testing of the model produces a significant effect, we can be more certain of the generalizability of the result.

3.3.1 Correlations

To create the model, correlations between all the templates and D-Level were computed. Of all the templates, two correlated significantly with D-Level: *-ing* complements such as the word *being* in the phrase *try being*, and verbs taking a finite complement, such as the verb *said*. Table 3.1 shows the p-values and r-values for each of the templates correlated significantly with D-Level.

Model	$p < 0.0400$	$F = 3.63(2, 27)$	$R = 0.212132$
Contributing Templates			
Template	$p <$	*Estimate*	*Error*
-ing Complement	0.2671	10.14985272	8.95752349
Finite Complement	0.1105	2.01346874	1.22003734
Intercept	0.0001	3.54656394	0.27207505

Table 3.2: Model 1: Parameters of the Model

3.3.2 Model

The templates mentioned above were used as factors in a multiple regression analysis using D-Level as a solution to an equation and the template values as possible values. Table 3.2 shows the p-value for the model in general as well as the p-values used in the model and the estimate of the coefficient for each template variable. Equation 3.3 shows the equation found to fit the data.

$$ShaC = 10.14985272x + 2.01346874y + 3.54656394 \tag{3.3}$$

The model found fit the data well overall accounting for 21.2% of the variance in D-Level scores. Neither template alone contributed significantly to the result. In Figure 3.1 the linear relationship between DLevel and ShaC score is apparent.

3.3.3 Test

Equation 3.3 was tested on 15 new data items. The values for *-ing* complement and finite complement were plugged into an equation which used the estimates in Table 3.2 as coefficients and a correlation between the resulting value and D-Level was computed. If the fit were good, a high correlation would be expected. The correlation was significant, but barely so. The result is summarized in the first row of Table 3.7, along with the results from the

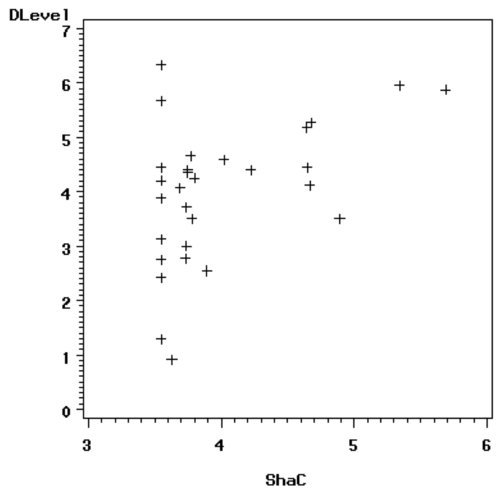

Figure 3.1: Correlation of Model 1 with D-Level (model sentences)

tests to come. The general linear relationship between D-Level and ShaC score is apparent in Figure 3.2.

3.3.4 Discussion

With a larger dataset one might be more confident of the result. However, the result does show promise. If it is a good result, the low p value might be attributed to the ordering of the data: the input to the model and to the test set were qualitatively different enough that the model was not a good fit for the test set. If this is the case then reordering the data and running the test again should yield a different result. It should be noted however that a significant result with the current ordering gives promise that the result is generalizable. As discussed above, the two data sets were not collected at the same time and contained some texts from different registers.

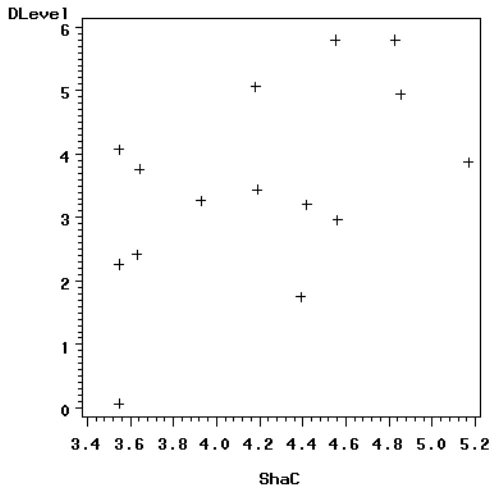

Figure 3.2: Correlation of Model 1 with D-Level (test sentences)

3.4 Model 2

This model divides the 45 data points into two roughly equal groups: a set to make the model ($n = 22$), comprised of the even numbered data points of the 45, and a set to test it ($n = 23$), comprised of the odd numbered data points. The intent was to mix the data together and produce a model and test set that were more homogenous. The expected result was a stronger model.

3.4.1 Correlations

To create a model correlations between all the templates and D-Level in the model set (n = 22) of data were computed. Of all the templates three correlated significantly with D-Level: verbs taking an infinitive complement (*try to go*), verbs taking a finite complement, and comparatives (*than*). In addition relative clauses like *the boy that the girl liked* produced an almost significant correlation, but were not included in the model. Table 3.3 shows the tem-

Template	$p <$	r
Infinitive Complement	0.0099	0.53756
Finite Complement	0.0121	0.52487
Comparatives	0.0138	0.51674
Relative Clauses	0.0510	0.42112

Table 3.3: Model 2: Templates Correlating Significantly with D-Level

plates which correlated significantly with D-Level along with their *p*-values and correlation coefficients.

3.4.2 Model

The variables presented in Table 3.3 were used as values in a multivariate equation. A multiple regression analysis was performed using these variables with D-Level as discussed in Section 3.2. Table 3.4 shows the *p*-value for the entire model. In addition, it shows the estimate of the coefficient for each template as well as its *p*-value. Equation 3.4 shows the equation found to fit the data.

$$ShaC = 4.611915015x + 1.80906668y + 4.844536568z + 2.725939942 \tag{3.4}$$

The resulting model fit the data well, accounting for 46.0% of the variation in D-Level scores. Only infinitive complements contributed significantly to the fit of the model. The goodness of fit of the model and the tight linear relationship between D-Level and ShaC score can be seen in Figure 3.3.

3.4.3 Test

The estimates for the coefficients of an equation described in Table 3.4 were used to create a new equation. Twenty-three new data points were used to solve this equation. The correlations between the solutions found and D-Level were computed. As expected of a good

Model	$p < 0.0099$	$F = 5.11(3, 18)$	$R = 0.459956$
Contributing Templates			
Template	$p <$	*Estimate*	*Error*
Infinitive Complement	0.01233	4.611915015	2.85273333
Finite Complement	0.1371	1.80906668	1.16261598
Comparatives	0.2353	4.844536568	3.94537399
Intercept	< 0.0001	2.725939942	0.49210236

Table 3.4: Model 2: Parameters of the Model

model, a high positive correlation was found. The result is summarized in the second row of Table 3.7, along with the other tests.

3.4.4 Discussion

The significance of the test shows a few things. First, as expected, when the data is mixed to be more homogenous, a better model can be created. This better model may also be attributed to the smaller model set and larger test set. It is difficult to determine which of these reasons explains the better result from the present experiment. More testing will shed more light.

Second, the variables that correlated with D-Level were different with the exception of verbs taking a finite complement. The other two were new. The difference is due probably to concentration of templates in each grouping. In Model 1 there were most likely more *-ing* complements, while there were more infinitive complements and comparatives in the second grouping. This presents an argument for increasing the size of the dataset considerably, and collecting data from more diverse sources. That the variables that correlated with D-Level were different also suggests that it will be difficult to determine exactly which templates drive the ranking of sentences. The hope was to find a small set of templates that definitively drove D-Level rankings. The preceding two experiments suggest a few candidates and point towards

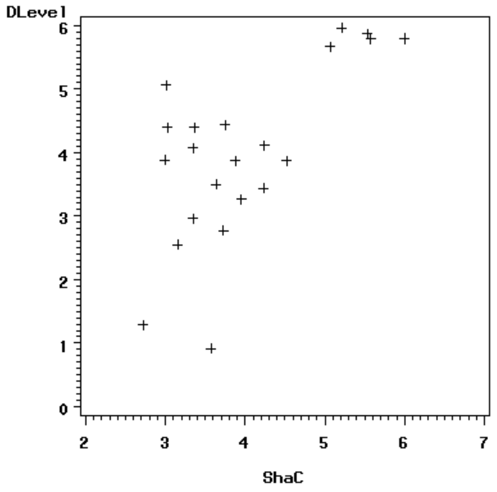

Figure 3.3: Correlation of Model 2 with D-Level (model sentences)

a general trend. We might look at verbs taking infinitive complements, *-ing* complements, and finite complements to get a very rough estimate of D-Level. The result implies that a portion of the "complexity" that D-Level keys in on is related directly to the acquisition of certain words and the ability to use them. It is possible that when infants acquire these structures they simply know how to use certain words but do not understand a general pattern, or that the words are the gateway to syntax.

3.5 Model 3

The results of the preceding model were due either to more homogenous data or to the fact that the model set was smaller and the test set larger. To determine the extent to which each of these is the case, the data was divided in yet a different way. Thirty-one data points from all the even numbered data as well as the first nine odd numbered data points in the 45 items comprised the model set. Fourteen data points from the odd numbered items were used as a test set.

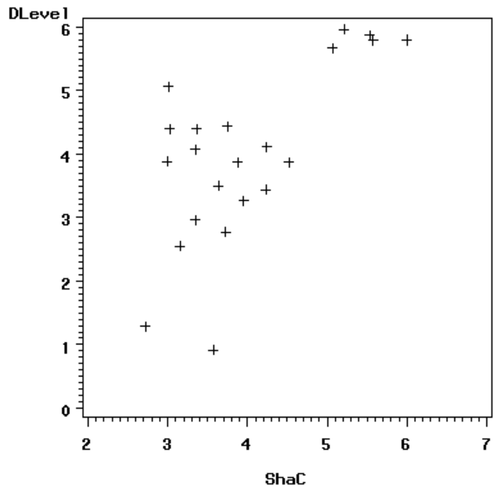

Figure 3.4: Correlation of Model 2 with D-Level (test sentences)

Template	$p <$	r
Infinitive Complement	0.0124	0.44381
Finite Complement	0.0007	0.57582
Comparatives	0.0106	0.45239

Table 3.5: Model 3: Templates Correlating Significantly with D-Level

3.5.1 Correlations

To create a model of the data, correlations between the templates and D-Level in the 31 data points in the first group were computed. Of all the templates, three correlated significantly with D-Level: verbs taking infinitive complements, verbs taking finite complements, and comparatives. Table 3.5 shows the templates that correlated significantly with D-Level along p-values and correlation coefficients.

Model	$p < 0.0011$	$F = 7.16(3, 27)$	$R = 0.443169$
Contributing Templates			
Template	$p <$	*Estimate*	*Error*
Infinitive Complement	0.2297	2.410039860	1.96101044
Finite Complement	0.0226	2.332029186	0.96402496
Comparatives	0.0815	5.249023585	2.90016873
Intercept	< 0.0001	2.918194863	0.36987544

Table 3.6: Model 3: Model

3.5.2 Model

The above variables were used as solutions to a multivariate equation. A multiple regression analysis was performed using these variables with D-Level to find coefficients to this equation. Table 3.6 shows the p-value found for the entire model. In addition it shows each template included in the model along with an estimate as to its coefficient and a p-value reflecting the strength of that template in the model. Equation 3.5 shows the equation found to fit the data.

$$ShaC = 2.410039860x + 2.332029186y + 5.249023585z + 2.918194863 \tag{3.5}$$

The resulting model fit the data well, accounting for 44.3% of the variance. In this case verbs taking a finite complement contributed significantly to the model. Neither of the other variables did. The intercept contributed significantly. Figure 3.5 shows a strong linear relationship between the variables.

3.5.3 Test

The estimates above for the coefficients of an equation that fit the data were put into an actual equation. Values from a new dataset ($n = 14$) were used to solve the equation. The correlation between these solutions and D-Level was computed. The result is shown in Table

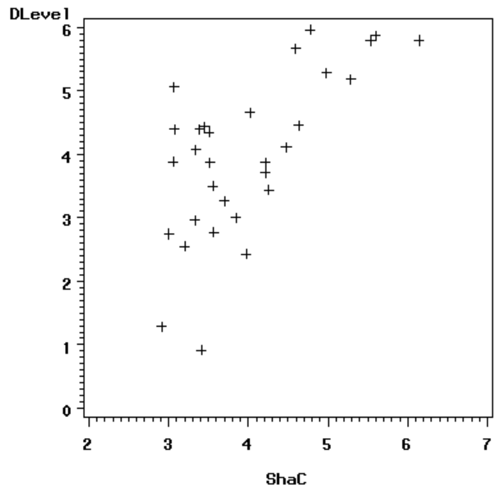

Figure 3.5: Correlation of Model 3 with D-Level (model sentences)

3.7. The strong positive correlation shows a good fit of the model to D-Level. Figure 3.6 shows a visualization of this trend.

3.5.4 Discussion

This model did not fit the test data as well as the previous model fit its test data. This suggests again that the data in the model and test sets was not homogenous. We do see an improvement in this model over the first model suggesting it is more homogenous. But again that the test portion produced a less significant result than in Model 2 suggests perhaps that the size of the test set needs to be increased to increase the significance of the result.

Of note is the fact that this model found the same templates to correlate significantly with D-Level as did Model 2. This suggests that at least in this dataset there are a few templates that do provide a good indication and that the speculations on picking out certain highly frequently used words as indicators of complexity may be a good shortcut to estimating D-Level.

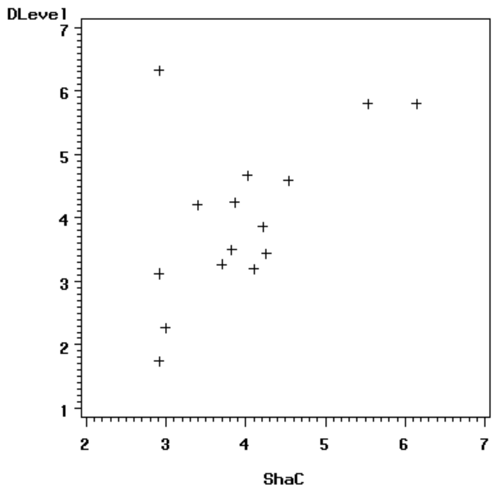

Figure 3.6: Correlation of Model 3 with D-Level (test sentences)

Model	n	$p <$	r
1	15	0.0450	0.52402
2	23	0.0069	0.54734
3	14	0.0381	0.55814

Table 3.7: Test Summary

3.6 General Discussion

The preceding three experiments confirm that the output of ShaC presented here does correlate significantly with D-Level, and that the correlation holds regardless of the way the data is organized. Three different ways of organizing the data produced significant results. Table 3.7 shows that of the three models, model 2 was best. This may be due to two factors. The data itself may have been more homogenous, since both datasets were mixed, and the larger test set may have provided a more reliable indication of the performance of the model. In either case, further testing on a larger dataset is in order.

ShaC is a good approximation of D-Level in spite of the fact that it makes errors. ShaC will still over generate, find strings that in some way match a template but are not an indicator of D-Level as one would rank by hand. Furthermore, there are instances of indicators of complexity that ShaC will not recognize. It does not analyze D-Level perfectly. However, those things that it misses may be included in sentences that are ranked level 7 by the D-Level scale. Thus, if it picks out two of five things contributing to a level 7 ranking, it has found enough to correlate with D-Level, since two or more indicators in the same sentence are always ranked as level 7 according to D-Level. Future improvements will only serve to strengthen its already solid approximation.

The three experiments centered on just a few templates that correlated most highly with D-Level. These were: *-ing* complements, infinitive complements, finite complements, comparatives, and some relative clauses. Only finite complements contributed to all three models, however infinitive complements and comparatives played a role in models 2 and 3. This suggests a benefit to testing on a larger dataset to determine exactly which templates contribute most to D-Level across a wide variety of registers. The problem is that different indicators of complexity are used in different registers, as noted above. Depending on the circumstances under which one is speaking, they will produce different indicators of complexity. This means that only with a large enough dataset to cover most of the different scenarios under which one might speak will a sense of which templates drive D-Level emerge. This knowledge would say something more about the way that English is used every day than anything about D-Level.[2]

[2]It should be clear that there is nothing inherent to D-Level that says it is an ideal indicator of complexity for any specific purpose. The fact that it works or might work to distinguish groups of speakers just means that it picks up on the right patterns in human speech. Thus the question of which templates contribute most to D-Level is of a passing interest until a larger scale analysis can be done.

Chapter 4

Conclusions

I have just scratched the surface of an implementation of the D-Level scale. There are numerous avenues that might be explored to improve upon ShaC as it stands. The following is an outline of future work.

4.1 Evaluation of Performance

In order to get a good sense of where improvements can be made, a realistic look at the performance of ShaC as it stands is in order. Three models showed the output of ShaC to correlate fairly well with the performance of the D-Level scale. This was in spite of looking at just a fraction of the sentence types covered by the D-Level. The sample in the experiment was small, and the results are probably not representative of ShaC's performance on larger data sets. In ShaC's favor, the texts analyzed came from sources which one might expect would have very high complexity, newspapers and academic texts, for example, and ShaC was still able to produce results that correlated well with D-Level. This is an indication that the templates that ShaC uses cover a sufficient amount of the D-Level scale to be able to rank most utterances with accuracy. While it will not give a full D-Level analysis it will do a good preliminary job, saving time on doing hand rankings. It will prove useful especially in conjunction with other methods of determining sentence complexity.

4.2 Improvements to ShaC

The lexicon is the core of a good D-Level analyzer. A major stumbling block for the present approach is lack of knowledge about verbs. If the lexicon was enriched to include verb argument structure, for example, the tendency for ShaC to find false positives would be reduced. There does not seem to be a good way to do this with the current lexicon. There is however promising research incorporating verb argument structures into parsers using a more robust tool set (Shi and Mihalcea, 2005). The approach would apply here as well.

Part of speech tagging may prove useful. As it stands, ShaC will make errors in gleaning what little part-of-speech information it does use from the words surrounding a target word. While part of speech taggers give tags that are not entirely useful, simply knowing whether something is some kind of noun or verb would be extremely useful. From this information, all of the same specialized categories of words used in the templates could be derived. Incorporating a part of speech tagger would mean some part of ShaC's job would have to be done in something other than Prolog. Having to use two systems instead of one may be seen as a disadvantage.

Part of speech tagging still does not broaden ShaC's coverage. Many of the D-Levels are left unranked. There may be ways to bridge this gap. Level 3 includes relative clauses modifying the object of the main verb. The template for these sentences is `n,comp,vpsed`. That something is in object position can be verified by broadening the template to `vpsed,n,comp,vpsed`. Level 6 includes relative clauses modifying the subject of the main verb. There is no template for these, but the level 3 template will work: `n,comp,vpsed`. This template generates all those structures that D-Level classifies as level 3 and all those that D-Level classifies as level 6. An estimate of the number of strings in a text matching a level 6 template can be computed as the difference between the number of strings matching the template `vpsed,n,comp,vpsed` and those matching the template `n,comp,vpsed`. There were no examples of a sentence with a relative clause modifying the subject in the data sets tested. This modification would then only improve ShaC's performance slightly.

4.3 Applications

This work was motivated by the potential for syntactic analysis to shed light on the nature of speech in the mentally ill. In particular syntactic complexity might be an indicator of working memory span. It is this idea of using syntactic complexity as a key to other information about a speaker that drives this project. The big open question is not whether ShaC mimics D-Level – it does to some extent – The question is whether ShaC will be useful in helping determine the syntactic characteristics of the speech of the mentally ill. It does not particularly matter if ShaC does or does not mimic D-Level so long as it picks up on useful trends in speech. As such, an open research question is can ShaC differentiate between normal and mentally ill speakers of English? If ShaC can distinguish these groups then it is picking up on something useful. If it cannot, and if the analysis of D-Level scale by Cheung and Kemper (1992) is correct, then we might deduce that the key to the D-Level analysis is really in the way it distinguishes between level 3 sentences and level 6 sentences. This distinction would amount to the difference between left and right branching sentences, and could be measured using other metrics. The result would invalidate the current approach, but would be very significant. It would imply that in reality working memory is the driving force behind some types of mental illness and any complexity metric that looks exclusively at those structures that tax working memory will be useful in making this distinction. Note however that if the data set summarized in Appendix B is any indication, level 6 sentences are rare even in written texts.

If it cannot, can it at least help to differentiate? Cheung and Kemper (1992) suggest D-Level in conjunction with Mean Clauses per Utterance (MCU) provides a good map of a speaker's syntactic complexity. MCU indicates depth of clausal embedding while D-Level indicates type of clausal embedding. Indeed the templates specify the exact form embedding takes. They suggest MCU is fairly easy to estimate. Simply count the number of subordinating conjunctions. It would be worth while to look at the interactions between MCU and D-Level in normal and mentally ill subjects.

Idea density (Kintsch and Keenan, 1973) is a measure of the number of propositions in a unit of text. The D-Level scale is in some sense the inverse of idea density.[1] Those things which D-Level rates, idea density ignores and vice versa. Cheung and Kemper (1992) suggest that a competent conversationalist will use simple syntax and presumably low idea density in order to be understood. Less competent speakers use more complicated syntax, but have lower idea density. This relationship has yet to be explored in the mentally ill.

4.4 Future Directions

Possibilities abound for future uses of ShaC. My hope is those outlined above and others will be explored. The need for good syntactic analysis tools is apparent. Only by being able to process large amounts of text data quickly and accurately will we be able to do larger scale experiments in schizophrenia and aphasia. And it is only with large data sets that we can get a good idea of the nature of the language of these subjects. As computers become faster and programs more sophisticated, we position ourselves to revolutionize psycholinguistic research along these lines. This thesis is a small step in that direction.

[1]I thank Dr. Michael Covington for solidifying this observation

Bibliography

Abney, S. (1987). *The English Noun Phrase in its Sentential Aspect*. PhD thesis, MIT.

Abney, S. (1991). Parsing by chunks. In *Principle-based Parsing*. Kluwer Academic Publishers.

Barch, D. and Berenbaum, H. (1994). The relationship between information processing and language production. *Journal of Abnormal Psychology*, 103:241–250.

Barch, D. and Berenbaum, H. (1997). Language generation in schizophrenia and mania: The relationships among verbosity, syntactic complexity, and pausing. *Journal of Psycholinguistic Research*, 26(4):408–412.

Biber, D., Conrad, S., and Reppen, R. (2000). *Corpus Linguistics: Investigating Language Structure and Use*. Cambridge University Press.

Brooks, P. (2003). *SCP: A Simple Chunk Parser*. University of Georgia. http://www.ai.uga.edu/mc/pronto.

Channell, R. W. (2003). Automated developmental sentence scoring using computerized profiling software. *American Journal of Speech-Language Pathology*, 12:369–375.

Cheung, H. and Kemper, S. (1992). Competing complexity metrics and adults' production of complex sentences. *Applied Psycholinguistics*, 13:53–76.

Covington, M. (2003). *ET: The Efficient Tokenizer*. University of Georgia. http://www.ai.uga.edu/mc/pronto.

Covington, M., He, C., Brown, C., Naci, L., and Brown, J. (2004). How complex is that sentence? A proposed revision of the Rosenberg and Abbeduto D-Level scale. manuscript.

De Juan, D. (2004). CCL: A closed-class lexicon for the ProNTo toolkit. Obtained through personal correspondence.

Frazier, L. (1985). Syntactic complexity. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, pages 129–189. Cambridge University Press, Cambridge.

Hunt, K. (1965). *Grammatical structures written at three grade level*. Number 3 in NTCE Research Report. National Council of Teachers of English, Champaign, Il.

Kemper, S., Greiner, L., Marquis, J., Prenovost, K., and Mitzner, T. (2001). Language decline across the life span: findings from the nun study. *Psychology and Aging*, 16(2):227–239.

Kemper, S., Kynette, D., Rash, S., Sprott, R., and O'Brien, K. (1989). Life-span changes to adults' language: Effects of memory and genre. *Applied Psycholinguistics*, 10:49–66.

King, K., Fraser, W., Thomas, P., and Kendell, R. (1990). Re-examination of the language of psychotic subjects. *British Journal of Psychiatry*, 149:211–215.

Kintsch, W. and Keenan, J. (1973). Reading rate and retention as a function of the number of propositions in the base structure of sentences. *Cognitive Psychology*, 5:257–274.

Lee, L. (1974). *Developmental sentence analysis: A grammatical assessment procedure for speech and language clinicians*. Northwestern University Press.

MacWhitney, B. (2002). *CLAN Manual*. Carnegie Mellon University. http://childes.psy.cmu.edu/pdf/clan.pdf.

Morice, R. (1985). Comprehension and production of complex syntax in schizophrenia. *Cortex*, 24(4):567–580.

Morice, R. and Ingram, J. (1983). Language, complexity, and age of onset of schizophrenia. *Psychiatry Research*, 9:233–242.

Rosenberg, S. and Abbeduto, L. (1987). Indicators of linguistic competence in the peer group conversational behavior of mildly retarded adults. *Applied Psycholinguistics*, 8:19–32.

Scarborough, H. (1990). Index of productive syntax. *Applied Psycholinguistics*, 11:1–22.

Schlachter, J. (2003). *ProNTo_Morph: Morphological Analysis Tool*. University of Georgia. http://www.ai.uga.edu/mc/pronto.

Shi, L. and Mihalcea, R. (2005). Putting the pieces together: Combining FrameNet, VerbNet, and WordNet for robust semantic parsing. In *Proceedings of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics*, Mexico.

Snowdon, D., Kemper, S., Mortimer, J., Greiner, L., Wekstein, D., and Markesbery, W. (1996). Linguistic ability in early life and cognitive function and Alzheimer's disease in late life: findings from the nun study. *Journal of the American Medical Association*, 275(7):528–532.

Thomas, P., Kearney, G., Napier, E., Ellis, E., Leudar, I., and Johnson, N. (1996a). The reliability and characteristics of the brief syntactic analysis. *British Journal of Psychiatry*, 168:334–337.

Thomas, P., Leudar, I., Napier, E., Kearney, G., Ellis, E., Ring, N., and Tantam, D. (1996b). Syntactic complexity and negative symptoms in first onset schizophrenia. *Cognitive Neuropsychiatry*, 1(3):191–200.

University, P. (2005). *WordNet 2.1*. http://wordnet.princeton.edu/.

Witzig, S. (2003). *Accessing WordNet from Prolog*. University of Georgia. http://www.ai.uga.edu/mc/pronto.

Yngve, V. (1960). A model and a hypothesis for language structure. *Proceedings of the American Philosophical Society*, 104:444–466.

Appendix A

The Revised D-Level Scale

For each level, the template used to detect that level is indicated at far right.

Level	Description	Example	Template
Level 0	Simple sentences including questions	*The dog barked.*	
		Did the dog bark?	
		Where are you going?	
	Sentences with auxiliaries and semi-auxiliaries	*This* may have *solved it.*	
		He is going to *take the bus.*	
	Simple eliptical (incomplete sentences)	*The dog over there.*	
		He did.	
Level 1	Infinitive or *-ing* complement with same subject as main clause	*Try* to bush her hair.	`vpsed,to,vpl`
		Try *brushing her hair.*	`vnfc,ving`
		I felt like *turning it.*	
Level 2	Conjoined noun phrases in subject position	John and Mary *left early.*	
	Sentences joined with a coordinating conjunction	*I brought candy* and *Peter cleaned up.*	

	Conjoined verbal, adjectival or adverbial constructions	*He* sang and jumped *on the way home.*	
Level 3	Relative (or appositional) clause modifying object of main verb	*The man scolded the boy* who stole the bicycle.	`n,comp,vpsed`
			`vpsed,n,comp`
		I like the girl who the boy likes.	`n,comp,n,vpsed`
	Nominalization in object position	*Why can't you understand* his rejection of the offer?	`pnd,nom`
	Finite clause as object of main verb	*John knew* that Mary was angry.	`vnfc,comp,n,v`
		Remember where it is.	`vpsed,comp,n,v`
		compv	
	Subject extraposition	It *was surprising* for John to have left Mary.	`it,bev,ving,comp2`
			`it,bev,adj,comp2`
			`it,bev,vpsed,comp2`
	Raising	John *seems to Mary* to be happy.	
Level 4	*-ing* form in complement (object) position	*He loves* visiting his grandfather.	
	Non-finite compement with its own understood subject	*I expect* him to go.	`vnfc,n,to,vpl`
		I want it done today.	`vnfc,nacc,vpl`
		I consider John a friend.	
		I want These animals out of my house.	

	Comparative with object of comparison	*John is* older than Mary.	`than`
		John is as old as Mary.	`as,adj,as`
Level 5	Sentences joined by a subordinating conjunction	*They will play today* if it does not rain.	`subconj`
	Non-finite clauses in adjunct (not complement) positions	*Cookie Monster touches Grover* after jumping over the fence.	`ping,ving`
		Having tried both, *I prefer the second one.*	
Level 6	Relative (or appositional) clause modifying subject of main verb	*The man* who cleans the rooms *left early today.*	
	Embedded clause serving as subject of main verb	For John to have left Mary *was surprising.*	
	Nominalization serving as subject of main verb	John's refusal of the drink *angered Mary.*	
Level 7	More than one kind of embedding in a single sentence	*John decided* to leave Mary when he heard that she was seeing Mark.	

The following are template symbols and their meanings for use with the above.

Symbol	Meaning
`vpsed`	verb with a plain, -*s*, or -*ed* ending
`vpl`	verb in plain form
`vnfc`	verb that takes a non-finite complement
`ving`	verb with an -*ing* ending
`vppl`	past participle of a verb
`v`	verb of any kind
`bev`	be verb
`conjs`	any conjunction
`nom`	nominalization
`n`	noun of any kind
`pnd`	pronoun that acts like a determiner
`comp`	complementizer
`adj`	adjective
`ping`	pronoun that can take an -*ing*
`subconj`	subordinating conjunction
`compv`	verb taking a finite complement

Appendix B

Verbs Taking a Non-Finite Complement

The following is a list of verbs that may take a non-finite complement.

```
word(vnfc,want).
word(vnfc,desire).
word(vnfc,believe).
word(vnfc,consider).
word(vnfc,insist).
word(vnfc,imagine).
word(vnfc,envisage).
word(vnfc,like).
word(vnfc,hate).
word(vnfc,love).
word(vnfc,dislike).
word(vnfc,have).
word(vnfc,order).
word(vnfc,demand).
word(vnfc,hear).
word(vnfc,see).
word(vnfc,feel).
word(vnfc,stop).
word(vnfc,promise).
word(vnfc,wish).
word(vnfc,keep).
word(vnfc,favor).
word(vnfc,fancy).
word(vnfc,admit).
word(vnfc,concede).
word(vnfc,prefer).
word(vnfc,adore).
word(vnfc,proclaim).
word(vnfc,report).
word(vnfc,observe).
word(vnfc,think).
word(vnfc,suppose).
word(vnfc,let).
word(vnfc,help).
word(vnfc,make).
word(vnfc,catch).
word(vnfc,mind).
word(vnfc,remember).
word(vnfc,recollect).
word(vnfc,stand).
word(vnfc,tollerate).
word(vnfc,resent).
word(vnfc,notice).
word(vnfc,get).
word(vnfc,keep).
word(vnfc,watch).
```

Appendix C

Overview of Hand-Ranked Texts

The table below summarizes the distribution of sentences of each D-Level in the analyzed texts. Total average score reflects the average score of all sentences across all texts. Average score for each text reflects the average sentence complexity solely for that text.

	Level									
Text	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Av. Score**	**Total Sentences**
newyork	9	0	0	3	1	3	0	17	4.45	33
harpers2	17	3	1	3	3	6	3	4	2.55	40
ctz	0	0	0	3	2	1	0	5	5.18	11
harpers1	7	1	1	3	2	2	0	12	4.07	28
csm	6	0	0	3	3	3	0	6	3.71	21
pan	12	1	0	0	0	1	1	7	2.77	22
gray	14	0	0	3	1	2	2	6	2.75	28
cristo	9	1	0	4	0	2	2	8	3.50	26
meta2	16	5	0	5	0	3	0	13	3.00	42
christie	17	3	0	2	0	0	1	1	0.92	24
bronte	4	0	0	3	0	1	1	16	5.28	25
sawyer	5	1	0	2	0	2	0	7	3.88	17
lesmis	4	1	0	3	0	3	1	8	4.35	20
cp	5	0	0	1	0	0	0	9	4.40	15
meta	6	3	0	1	0	0	0	4	2.43	14
dlevel1	3	0	0	1	0	0	0	4	3.88	8

dlevel2	1	0	0	3	0	1	0	4	4.67	9
dlevel3	0	0	0	1	0	2	0	3	5.67	6
dlevel4	3	0	0	2	0	0	2	1	3.13	8
dlevel5	2	0	0	3	0	0	0	5	4.40	10
dlevel6	1	1	0	0	0	0	1	2	4.20	5
dlevel7	2	1	0	0	0	1	1	4	4.44	9
dlevel8	2	0	0	2	0	0	0	4	4.25	8
dlevel9	5	0	0	1	0	0	1	0	1.29	7
dlevel10	0	0	0	0	0	1	0	2	6.33	3
nd	2	0	0	1	0	0	0	13	5.88	16
reuters	4	0	1	1	0	2	0	9	4.59	17
sfc	2	0	1	1	1	1	0	21	5.96	27
voa	4	0	1	6	0	3	0	4	3.50	18
abcnews	2	0	0	3	0	0	0	4	4.11	9
addison	5	0	0	2	0	2	0	6	3.87	15
dlevelex	7	3	0	6	7	3	3	1	2.97	30
ajc	0	0	0	3	0	0	0	7	5.80	10
alger2	10	2	0	5	2	3	0	8	3.20	30
charniak	2	0	0	2	0	4	1	7	5.06	16
dahl	14	0	0	2	0	2	0	6	2.42	24
kids	2	0	0	4	1	4	0	3	4.07	14
nytimes	0	1	0	1	0	2	0	7	5.73	11
schizdoc	5	2	0	3	1	3	0	7	3.76	21
dana	8	0	0	0	1	2	0	1	1.75	12
electricity	17	0	0	0	0	11	1	1	2.27	30
cnn	3	1	0	3	0	1	0	12	4.95	20
dearabby	8	0	4	1	3	0	0	9	3.44	25

kid3yr	27	2	0	0	0	0	0	0	0.07	29
rtest	14	0	3	4	0	4	1	11	3.27	37
Totals	286	32	12	100	28	81	22	289	3.56	850

Appendix D

Program

```
:- multifile word/2.


:- ensure_loaded('scp2.pl').
:- ensure_loaded('pronto_morph_engine.pl').
:- ensure_loaded('cclalt.pl').
:- ensure_loaded('lexicon2.gmr').
:- ensure_loaded('rules.gmr').
:- ensure_loaded('et.pl').



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Predicates that call the FSA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% rank a single sentence from all appropriate templates
rank_a_sentence(String,Templates) :-
    get_patterns(Templates,Patterns),
    get_children,
    rank_lists(Patterns,[String]).

rank_sentence_file(File,Templates,OutFile) :-
    tell(OutFile),
    rank_sentences(File,Templates),
    told.

% rank all the sentences in a set of
% files and output the results to a file.
rank_sentences_files(TempFile,Templates,OutFile) :-
   ensure_loaded(TempFile),
   get_patterns(Templates,Patterns),
   get_children,
   tell(OutFile),
   rank_sentences_files_aux(Patterns,1),
   told.

rank_sentences_files_aux(Patterns,Num) :-
   file_loc(Num,FileName),
   open(FileName,read,Stream),
   get_lists(Stream,Lists),
   close(Stream),
   write('file: '),write(FileName),nl,
   rank_lists(Patterns,Lists),
   Num2 is Num+1,
   rank_sentences_files_aux(Patterns,Num2).

rank_sentences_files_aux(_,_).

% rank each line of a single file separately.
% returns all of the rankings it found at each level
rank_sentences(FileName,Templates) :-
   get_patterns(Templates,Patterns),
   get_children,
   open(FileName,read,Stream),
   get_lists(Stream,Lists),
   close(Stream),
   rank_lists(Patterns,Lists).

get_lists(Stream,[]) :-
   peek_code(Stream,X),
   X == -1.

get_lists(Stream,[Words|Lists]) :-
   tokenize_line(Stream,Line),
   tokens_words(Line,Words),
   get_lists(Stream,Lists).

rank_lists(_,[]) :- nl,nl,nl,nl,nl.


% Input is a list of Patterns and a List of items to be ranked.
% Ouput is a printed list of the ranks of the items
rank_lists(Patterns,[H|Tail]) :-
   write('sentence: '),write(H),nl,
   rank_text_aux_alt(Patterns,H,0,TotalScore,ScoreList),
   write('line total score: '),write(TotalScore),nl,
   print_scores(ScoreList),
   rank_lists(Patterns,Tail).

print_scores([]).

print_scores([(Level,Score)|Tail]) :-
   write('score level: '),write(Level),write(' score: '),write(Score),nl,
   print_scores(Tail).

rank_files_to_sas(TempFile,TemplateFile,OutFile) :-
    get_patterns(TemplateFile,Patterns),
    ensure_loaded(TempFile),
    tell(OutFile),
    rank_files_to_sas_aux(1,Patterns).
    told.

rank_files_to_sas_aux(Num,Patterns) :-
    file_loc(Num,1,FileName),
    write(FileName),nl,
    rank_text_sas(FileName,Patterns,0,_),
    Num2 is Num + 1,
    rank_files_to_sas_aux(Num2,Patterns).

rank_files_to_sas_aux(Num,Patterns) :-
    file_loc(Num,0,_),
    Num2 is Num + 1,
    rank_files_to_sas_aux(Num2,Patterns).

rank_files_to_sas_aux(Num,Patterns) :-
    Num < 50,
    Num2 is Num + 1,
    rank_files_to_sas_aux(Num2,Patterns).

rank_files_to_sas_aux(_,_).

rank_files_to_file(TempFile,TemplateFile,OutFile) :-
   tell(OutFile),
   get_patterns(TemplateFile,Patterns),
    ensure_loaded(TempFile),
   rank_files_to_file_aux(1,Patterns),
   told.
```

```
rank_files_to_file_aux(Num,Patterns) :-
    file_loc(Num,1,FileName),
   write('file: '),write(FileName),nl,
   rank_text_alt(FileName,Patterns,0,Score),
   write('total score: '),write(Score),nl,
   Num2 is Num + 1,
   rank_files_to_file_aux(Num2,Patterns).

rank_files_to_file_aux(Num,Patterns) :-
    file_loc(Num,0,_),
    Num2 is Num + 1,
    rank_files_to_file_aux(Num2,Patterns).

rank_files_to_file_aux(Num,Patterns) :-
    Num < 50,
    Num2 is Num + 1,
    rank_files_to_file_aux(Num2,Patterns).

rank_files_to_file_aux(_,_).


rank_from_templates_score_list(FileNum,TempFile,TemplateFile,Score) :-
   get_patterns(TemplateFile,Patterns),
   get_file_location(FileNum,TempFile,FileName),
   rank_text_alt(FileName,Patterns,0,Score).

% like rank_from_templates but accesses the files in a different way.
% uses a file template.
rank_from_templates(FileNum,TempFile,TemplateFile,Score) :-
   get_patterns(TemplateFile,Patterns),
   get_file_location(FileNum,TempFile,FileName),
   rank_text(FileName,Patterns,0,Score).

% consult a template file for the patterns used to rank the text
rank_from_templates(FileName,TemplateFile,Score) :-
   get_patterns(TemplateFile,Patterns),
   rank_text(FileName,Patterns,0,Score).


% checks to see if a list of atoms matches the given template.
rank_atoms(List,Template,Score) :-
   get_children,
   run_fsa(Template,List,0,Score).

% get_patterns from a file of template/2 clauses
get_patterns(FileName,PatternList) :-
   ensure_loaded(FileName),
   findall((X,Y),template(X,Y),PatternList).

get_file_location(Num,FileName,FileLocation) :-
   ensure_loaded(FileName),
   file_loc(Num,1,FileLocation).


rank_single_level(FileNum,FilesFile,Template,Score) :-
   get_file_location(FileNum,FilesFile,FileName),
   rank_text(FileName,[(1,Template)],0,Score).


% finds all occurrences of a given template in the given file
rank_single_level(FileName,Template,Score) :-
   tokenize_file(FileName,TokenList),
   tokens_words(TokenList,WordList),
   get_children,
   run_fsa(Template,WordList,0,Score).
```

```
% determines syntactic comlexity of a text by running through a
% file and finding all occurrences of all the provided templates and
% returns a score
rank_text(FileName,[(Level,TemplateHead)|Tail],Score,TotalScore) :-
   tokenize_file(FileName,TokenList),
   tokens_words(TokenList,WordList),
   get_children,
   rank_text_aux([(Level,TemplateHead)|Tail],WordList,Score,TScore),
   length(WordList,Length),
   TotalScore is TScore/Length.
% score is the raw score over the number of words in the text.

rank_text_aux([],_,Score,Score) :- !.

rank_text_aux([(Level,TemplateHead)|Tail],WordList,Score,TotalScore) :-
   write('level: '),write(Level),nl,write('template: '),write(TemplateHead),nl,
   run_fsa(TemplateHead,WordList,0,S),
   NewScore is Score+(S*Level),
   rank_text_aux(Tail,WordList,NewScore,TotalScore).

% ranks a text and outputs results in SAS readable format
rank_text_sas(FileName,[(Level,TemplateHead)|Tail],Score,TotalScore) :-
   tokenize_file(FileName,TokenList),
   tokens_words(TokenList,WordList),
   get_children,
   rank_text_sas_aux_alt([(Level,TemplateHead)|Tail],WordList,Score,TScore,ScoreList),
   length(WordList,Length),
   TotalScore is TScore/Length,
    print_score_list_sas(Length,ScoreList),
    write(TotalScore),nl.
% score is the raw score over the number of words in the text.

rank_text_sas_aux_alt([],_,Score,Score,[]) :- !.

rank_text_sas_aux_alt([(Level,TemplateHead)|Tail],WordList,Score,TScore,[(L2,S)|Rest]) :-
    run_fsa_alt(TemplateHead,WordList,0,S),
    NewScore is Score + (S*Level),
    L2= Level,
    rank_text_sas_aux_alt(Tail,WordList,NewScore,TScore,Rest).

print_score_list_sas(_,[]).

print_score_list_sas(Length,[(Level,Score)|Tail]) :-
   S is (Level*Score)/Length,
   write(S),write(' '),
   print_score_list_sas(Length,Tail).




% gets a list of the scores of each level and prints it out.
% scores for each level are calculated the same as scores for the whole text.
rank_text_alt(FileName,[(Level,TemplateHead)|Tail],Score,TotalScore) :-
   tokenize_file(FileName,TokenList),
   tokens_words(TokenList,WordList),
   get_children,
   rank_text_aux([(Level,TemplateHead)|Tail],WordList,Score,TScore,ScoreList),
   length(WordList,Length),
   TotalScore is TScore/Length,
   print_score_list(Length,ScoreList).



print_score_list(_,[]).

print_score_list(Length,[(Level,Score)|Tail]) :-
   S is (Level*Score)/Length,
   write('score level '),write(Level),write(': '),write(S),nl,
   print_score_list(Length,Tail).



rank_text_aux_alt([],_,Score,Score,[]) :- !.



rank_text_aux_alt([(Level,TemplateHead)|Tail],WordList,Score,TotalScore,[(Level,S)|Rest]) :-
   run_fsa(TemplateHead,WordList,0,S),
   % write('run fsa score: '),write(S),nl,
   S \== 0,!,
   write('level: '),write(Level),nl,write('template: '),write(TemplateHead),nl,
   NewScore is Score+(S*Level),
   rank_text_aux_alt(Tail,WordList,NewScore,TotalScore,Rest).

rank_text_aux_alt([_|Tail],WordList,Score,TotalScore,Rest) :-
   !,
   rank_text_aux_alt(Tail,WordList,Score,TotalScore,Rest).


% an alternate version of rank text aux that returns a list of scores at each level
rank_text_aux([],_,Score,Score,[]) :- !.

rank_text_aux([(Level,TemplateHead)|Tail],WordList,Score,TotalScore,[(Level,S)|Rest]) :-
   write('level: '),write(Level),nl,write('template: '),write(TemplateHead),nl,
   run_fsa(TemplateHead,WordList,0,S),
   NewScore is Score+(S*Level),
   rank_text_aux(Tail,WordList,NewScore,TotalScore,Rest).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rank all the sentences from a series of files and print the results in a sas-ready format
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rank_sentences_sas(FileName,Templates,OutFile) :-
   get_patterns(Templates,Patterns),
   get_children,
    ensure_loaded(FileName),
    tell(OutFile),
   rank_sentences_sas_aux(1,Patterns),
    told.

/*
rssa(Num,FileName,Templates) :-
    get_patterns(Templates,Patterns),
    get_children,
    ensure_loaded(FileName),
    file_loc(Num,File),
    open(File,read,Stream),
    get_lists(Stream,Lists),
    close(Stream),!,
    write('* '),write(FileName),nl,
    rank_lists_sas(Patterns,Lists).
*/

rank_sentences_sas_aux(Count,Patterns) :-
    file_loc(Count,1,FileName),
    open(FileName,read,Stream),
    get_lists(Stream,Lists),
    close(Stream),
    write('* '),write(FileName),nl,
    rank_lists_sas(Patterns,Lists),!,
    C2 is Count + 1,
    rank_sentences_sas_aux(C2,Patterns).

rank_sentences_sas_aux(Count,Patterns) :-
    file_loc(Count,0,_),
    C2 is Count + 1,
    rank_sentences_sas_aux(C2,Patterns).

rank_sentences_sas_aux(Count,Patterns) :-
    Count < 30,
    C2 is Count + 1,
    write('not to be ranked'),nl,
    rank_sentences_sas_aux(C2,Patterns).

rank_sentences_sas_aux(_,_) :- !.

rank_lists_sas(_,[]) :- !.


% Input is a list of Patterns and a List of items to be ranked.
% Ouput is a printed list of the ranks of the items
rank_lists_sas(Patterns,[H|Tail]) :-
   length(H,Len),
   Len > 0,
   % write(H),nl,
   write(Len),write(' '),!,
   rank_text_sas_aux(Patterns,H,ScoreList),
   print_scores_sas(ScoreList,Len),nl,
   rank_lists_sas(Patterns,Tail).

rank_lists_sas(Patterns,[H|Tail]) :-
   H == 0,!,
   rank_lists_sas(Patterns,Tail).

rank_lists_sas(_,_) :-
    nl,write('failed to rank text or print scores'),nl.

% print the scores of each structure found in a sentence
print_scores_sas([],_) :- !.

print_scores_sas([(_,Count)|T],Len) :-
    Score is Count,


    write(Score),write(' '),
    print_scores_sas(T,Len).

/*
print_scores_sas([(_,Count)|T]) :-
    Count == 0,
    write(0),write(' '),
    print_scores_sas(T).
*/
pss_alt(_,0).

pss_alt(Level,Count) :-
    write(Level),write(' '),
    C2 is Count - 1,
    pss_alt(Level,C2).




% collect a list of levels and how many of each there are
rank_text_sas_aux([],_,[]) :- !.

rank_text_sas_aux([(Level,TemplateHead)|Tail],WordList,[(Level,S)|Rest]) :-
    run_fsa_alt(TemplateHead,WordList,0,S),
    rank_text_sas_aux(Tail,WordList,Rest).

run_fsa_alt(_,[],Score,Score) :- !.

% runs through the FSA. returns score for one level
```

```
run_fsa_alt(Pattern,String,Score,TotalScore) :-
    fsa(Pattern,String,Rest,S,_),!,
    NewScore is Score + S,
    run_fsa_alt(Pattern,Rest,NewScore,TotalScore).







% resets and runs the fsa every time a part of
% the string matching the given pattern is found.
run_fsa(_,[],Score,Score) :- !.

run_fsa(Pattern,String,Score,TotalScore) :-
   fsa(Pattern,String,Rest,S,Phrase),!,
   NewScore is Score + S,
   ((S == 1,
   write('phrase: '),write(Phrase),write(' '),
   write('count: '),write(NewScore),nl)
   ;
   true),
   run_fsa(Pattern,Rest,NewScore,TotalScore).

% run_fsa(_,_,Score,Score).






%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% the fsa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% this is a redo of the FSA portion of my thesis.
% It makes several improvements:
% - it only chunk parses prepositional, adverbial, and adjectival phrases
% - it skips over those phrases
% - it checks for patterns of words instead of phrases
% - some words must immediately be followed by others.
% some words can have phrases between them.


:- dynamic child_of/2.
% fsa(+Points,+Pattern,+String,-Rest,+Phrase,-CompletePhrase)
% accepts a pattern and a string of text.  Succeeds if the
% Pattern is found in the String. Returns the
% Rest of the String. Returns a 1
% to indicate a successful find or a 0 to
% indicate an unsuccessful find.
% fsa returns the string it found upon
% completing an analysis (CompletePhrase)

% no more Pattern.  Return what's left of the String.
fsa([],String,String,1,[]) :- !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% beginning of the string rules
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/*
% the current word in the input stream is an adjective.
% Parse a chunk and move past it.
fsa([-|T],[Adj|Tail],Rest,Score,[[Adj2]|Phrase]) :-
   (word(adj,Adj)
   ;
   (word(Cat,Adj),
   child_of(Cat,adjp))),!,
   Adj2= Adj,
   fsa(T,Tail,Rest,Score,Phrase).

% the current word in the input stream is an adverb.
% Parse a chunk and move past it.
fsa([-|T],[Adv|Tail],Rest,Score,[[Adv2]|Phrase]) :-
   (word(adv,Adv)
   ;
   (word(Cat,Adv),
   child_of(Cat,advp))),!,
   Adv2= Adv,
   fsa(T,Tail,Rest,Score,Phrase).
*/




%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% find particular word
%%%%%%%%%%%%%%%%%%%%%%%%%%%
fsa([Word|T],[Word|Tail],Rest,Score,[W|Phrase]) :-
   W= Word,!,
   fsa(T,Tail,Rest,Score,Phrase).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% beginning of phrase special cases
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fsa([-,Word|T],[Word|Tail],Rest,Score,[W|Phrase]) :-
   W= Word,!,
   fsa(T,Tail,Rest,Score,Phrase).

/*
fsa([-,Word|_],[W|Tail],Tail,0,[]) :-
   word(_,Word),
   Word \== W.
*/
% beginning of phrase skip a prepositional phrase.
% Since its at the beginning don't add it in.
fsa([-|T],[pp(_)|Tail],Rest,Score,Phrase) :-
   !,
   fsa([-|T],Tail,Rest,Score,Phrase).

fsa([-,subconj|_],[for,a,while|Tail],Tail,0,[]).

fsa([-,subconj|_],[for,example|Tail],Tail,0,[]).


fsa([-,subconj|_],[for,sure|Tail],Tail,0,[]).

fsa([-,subconj|_],[for,certain|Tail],Tail,0,[]).

fsa([-,subconj|_],[for,the,time,being|Tail],Tail,0,[]).

fsa([-,subconj|_],[as,normal|Tail],Tail,0,[]).

fsa([-,subconj|_],[as,usual|Tail],Tail,0,[]).

fsa([-,subconj|_],[as,per,usual|Tail],Tail,0,[]).


% there are execptions to this
fsa([-,subconj_],[and,so,on|Tail],Tail,0,[]).

% there are many exceptions to this pattern
```

```
fsa([-,subconj|_],[like,that|Tail],Tail,0,[]).



fsa([-,subconj|_],[Adj1,Adj2|Tail],Tail,0,[]) :-
   word(adj,[Adj1,Adj2]).

fsa([-,subconj|_],[Word,that|Tail],Tail,0,[]) :-
   member(Word,[such,like]).

fsa([-,subjconj|_],[that,is|Tail],Tail,0,[]).

fsa([-,subconj|_],[as,Adj,as|Tail],Tail,0,[]) :-
   word(adj,Adj).


% tries to handle 'in case of outbreaks' and similar sentences
% misses 'in case of all
fsa([-,subconj|_],[W1,W2,P|Tail],Tail,0,[]) :-
   word(subconj,[W1,W2]),
   word(p,P).


/*
% quick and dirty fix mostly for the verb 'have'
fsa([-,vnfc|_],[V1,V2|Tail],Tail,0,[]) :-
   word(vnfc,V1),
   word(adj,V2),
   word(vppl,V2).
*/

% quick and dirty fix mostly for the verb 'have'
fsa([-,vnfc|_],[V1,V2|Tail],Tail,0,[]) :-
   word(vnfc,V1),
   word(adj,V2),
   word(vpsed,V2).

% fsa wants vpsed.  the current word
% is a noun. the next word is a candidate,
% but has a noun form.  assume the second word is a noun.
fsa([-,vnfc|_],[N1,N2|Tail],Tail,0,[]) :-
   word(n,N1),
   \+ word(npl,N1),
   word(vnfc,N2),
   word(n,N2).

% fsa wants a vpsed. the current word is a
% candidate. the next word is
% an auxiliary verb.  assume the current
% word isn't a verb.
fsa([-,vnfc|_],[V1,V2|T],[V2|T],0,[]) :-
   word(vnfc,V1),
   word(vaux,V2),
   word(X,V1),
   child_of(X,dp).


% beginning of phrase
% fsa wants a vpsed. the word before a candidate verb is a determiner.
% assume candidate is not a verb. quit.
% trouble sentence: "this used to be..." "that seems to be..." "
fsa([-,vnfc|_],[V0,V1|T],T,0,[]) :-
   word(d,V0),
   \+ word(pronoun,V0),
   word(vnfc,V1),
   \+ word(adj,V1),!.
```

```
% fsa wants a vpsed. the word before a candidate verb is a determiner.
% assume candidate is not a verb. quit.
fsa([-,vnfc|_],[V0,V1|T],T,0,[]) :-
   word(adj,V0),
   \+ word(adv,V0),
   word(vnfc,V1),
   \+ word(adj,V1),!.

% if you have a preposition followed by a verb in -ing form,
% don't parse the preposition, instead accept the verb
fsa([-,vnfc|T],[P,V|Tail],Rest,Score,[V2|Phrase]) :-
   word(p,P),
   word(vnfc,V),
   word(ving,V),!,
   V2= V,
   fsa(T,Tail,Rest,Score,Phrase).

% if you are looking for a verb and see an "s" with
% two nouns around it it's not a verb, its a possessive
% skip onto the next parts
fsa([-,vpsed|T],[W1,s,W2|Tail],Rest,Score,Phrase) :-
   word(n,W1),
   word(n,W2),
   fsa([-,vpsed|T],Tail,Rest,Score,Phrase).

% fsa wants vpsed.  the current word is a noun.
% the next word is a candidate, but has a noun form.
% assume the second word is a noun.
fsa([-,vpsed|_],[N1,N2,N3|Tail],Tail,0,[]) :-
   word(n,N1),
   word(vpsed,N2),
   word(n,N2),
   N3 \== to.

% fsa wants a vpsed. the current word is a candidate.
% the next word is an auxiliary verb.
% assume the current word isn't a verb.
fsa([-,vpsed|_],[V1,V2|T],[V2|T],0,[]) :-
   word(vpsed,V1),
   word(vaux,V2),
   word(X,V1),
   child_of(X,dp).


% beginning of phrase
% fsa wants a vpsed. the word befor a candidate verb is a determiner.
% assume candidate is not a verb. quit.
% trouble sentence: "this used to be..." "that seems to be..." "
fsa([-,vpsed|_],[V0,V1|T],T,0,[]) :-
   word(d,V0),
   \+ word(pronoun,V0),
   word(vpsed,V1),
   \+ word(adj,V1),!.


% fsa wants a vpsed. the word before a candidate verb is a determiner.
% assume candidate is not a verb. quit.
fsa([-,vpsed|_],[V0,V1|T],T,0,[]) :-
   word(adj,V0),
   word(vpsed,V1),
   \+ word(adj,V1),!.

% you are at the beginning of the stream.
% fsa wants a vpsed.  accept if the next word is not an auxillary
% verb and the previous word isn't a determiner
fsa([-,vpsed|T],[V0,V1,V2|Tail],Rest,Score,[V3|T1]) :-
   \+ word(d,V0),
   \+ word(pronoun,V0),
   \+ word(p,V0),
   word(vpsed,V1),
   \+ word(vaux,V2),
   \+ word(vpsed,V2),
   !,
   V3= V1,
   fsa(T,[V2|Tail],Rest,Score,T1).

% fsa looking for a noun.  if the current word is a noun but not
% a pronoun, and the next word is also part of a noun phase, skip it.
fsa([-,n|T],[N1,N2|Tail],Rest,Score,[[N3]|Phrase]) :-
   N1 \== that,
   N2 \== that,
   word(C1,N1),
   C1 \== pronoun,
   child_of(C1,dp),
   word(C2,N2),
   C2 \== d,
   \+ word(p,N2),
   child_of(C2,dp),
   !,
   N3= N1,
   fsa([-,n|T],[N2|Tail],Rest,Score,Phrase).

% beginning of pattern
% skip determiners
fsa([-,n|T],[D|Tail],Rest,Score,[[D1]|Phrase]) :-
   word(d,D),
   \+ word(n,D),
   D1= D,
   !,
   fsa([-,n|T],Tail,Rest,Score,Phrase).

% at beginning of pattern
% accept a pronoun as a noun if the next word is not part of a noun phrase.
fsa([-,n|T],[Pr|Tail],Rest,Score,[Pr2|Phrase]) :-
   word(pronoun,Pr),
   Pr2= Pr,
   fsa(T,Tail,Rest,Score,Phrase).

fsa([-,X|T],[W1,W2,W3|Tail],Rest,Score,[W4,W5,W6|Phrase]) :-
   find_element(X,[W1,W2,W3]),!,
   W4= W1,
   W5= W2,
   W6= W3,
   fsa(T,Tail,Rest,Score,Phrase).

fsa([-,X|T],[W1,W2|Tail],Rest,Score,[W3,W4|Phrase]) :-
   find_element(X,[W1,W2]),!,
   W3= W1,
   W4= W2,
   fsa(T,Tail,Rest,Score,Phrase).


% the fsa is at the beginning at the current element
% in the stream doesn't match what the fsa is looking for
fsa([-,E|T],[H|Tail],Rest,Score,[H2|Phrase]) :-
   find_element(E,H),
   H2= H,!,
   fsa(T,Tail,Rest,Score,Phrase).

% The current word is a pp. it is followed
% by a determiner or a plural noun. skip the pp.
fsa([-|T],[PP,D|Tail],Rest,Score,Phrase) :-
   word(p,PP),
   (word(d,D)
   ;
   (PP \== to,
   word(n,D))
   ;
   (PP == to,
   \+ word(v,D))
   ;
   word(npl,D)
   ),
   parse(pp,[PP,D|Tail],Remain,Words,_),
   fsa([-|T],[pp(Words)|Remain],Rest,Score,Phrase).

% didn't find the initial element
fsa([-,E|T],[_|Tail],Rest,Score,Phrase) :-
   !,
   fsa([-,E|T],Tail,Rest,Score,Phrase).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% comp rules
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fsa([comp|_],[Comp,s|Tail],Tail,0,[]) :-
   word(comp,Comp).

% if you see a comp followed by
% a singular noun  assume the comp is a determiner and quit
% this makes errors: ie 'i think that fish is a tasty food'
% MontyTagger also makes this mistake
fsa([comp|_],[Comp,N|Tail],Tail,0,[]) :-
    word(comp,Comp),
    word(d,Comp),
    word(nsg,N),
    \+ word(pronoun,N).




%%%%%%%%%%%%%%%%%%%%%%%
% rules for skipping pps
%%%%%%%%%%%%%%%%%%%%%%%

% don't allow pp before a comp
fsa([comp|_],[pp([P|_]),W2|Tail],Tail,0,[pp,W3]) :-
   P \== of,
   word(comp,W2),!,
   W3= W2.

% skip pps otherwise
fsa(Pattern,[pp(W)|Tail],Rest,Score,[Words|T]) :-
   !,
   Words= W,
   fsa(Pattern,Tail,Rest,Score,T).

% special case for 'so that'.  real clumsy but i want it fixed now.
fsa([comp|_],[so,that|Tail],Tail,0,[]).

fsa([comp|T],[W1,W2|Tail],Rest,Score,[W3,W4|Phrase]) :-
    word(comp,[W1,W2]),
    W3= W1,
    W4= W2,
    !,
    fsa(T,Tail,Rest,Score,Phrase).


% looking for a comp.  see an adjective before a comp.  quit.
fsa([comp|_],[W1,W2|Tail],Tail,0,[]) :-
   word(adj,W1),
   word(comp,W2).


/*
fsa([comp|T],[W1,W2|Tail],Rest,Score,[W3|T1]) :-
   word(C1,W1),
   C1 \== adj,
   word(comp,W2),!,
   W3= W2,
   fsa(T,Tail,Rest,Score,T1).
*/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rules for skipping adjps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/*
% fail if you get an adjective that's not an ing verb after a be verb
fsa(_,[W1,W2|T],T,0,[W3]) :-
   W1 \== be,
   word(bev,W1),
   word(adj,W2),
   \+ word(ving,W2),
   !,
   W3= W1.
*/

%%%%%%%%%%%%%%%%%%%%%
% verb special cases
%%%%%%%%%%%%%%%%%%%%%

fsa([to,vpl],[Adj,to,V1|T],T,0,[]) :-
   word(adj,Adj),
   word(vpl,V1).


% don't accept an adjective before an ing verb
fsa([ving|T],[Adj,V|T],T,0,[]) :-
   word(adj,Adj),
   word(ving,V).

% some adjective s are also determiners.  don't skip these.
% if it is not an adjective, quit
fsa([vpsed|_],[Adj,Word|Tail],Tail,0,[]) :-
   word(adj,Adj),
   word(d,Adj),
   word(vpsed,Word),
   word(n,Word).

% some adjective s are also determiners.  don't skip these.
% if it is not an adjective, quit
fsa([ving|_],[Adj,Word|Tail],Tail,0,[]) :-
   word(adj,Adj),
   word(d,Adj),
   word(ving,Word),
   word(n,Word).


% if you are looking for a verb and see an "s"
% with two nouns around it it's not a verb, its a possessive
fsa([vnfc|_],[W1,s,W2|T],T,0,[]) :-
   word(n,W1),
   word(n,W2).

fsa([vpsed|_],[W1,s,W2|T],T,0,[]) :-
   word(n,W1),
   word(n,W2).

fsa([ving|_],[W1,s,W2|T],T,0,[]) :-
```

```
   word(n,W1),
   word(n,W2).

% fsa wants a vpsed. the current word is a candidate.
% the next word is an auxiliary verb.  assume the current word isn't a verb.
fsa([vpsed|_],[V1,V2|T],T,0,[]) :-
   word(vpsed,V1),
   word(vaux,V2),
   word(X,V1),
   child_of(X,dp).

% fsa wants a vpsed. the word before a candidate
% verb is a determiner. assume candidate is not a verb. quit.
fsa([vpsed|_],[V0,V1|T],T,0,[]) :-
   word(d,V0),
   word(vpsed,V1),!.


% fsa wants a vpsed. the word before a candidate
% verb is a determiner. assume candidate is not a verb. quit.
fsa([vpsed|_],[V0,V1|T],T,0,[]) :-
   word(adj,V0),
   word(vpsed,V1),!.


% if you have a preposition followed by a verb in -ing
% form, don't parse the preposition, instead accept the verb
fsa([vnfc|T],[P,V|Tail],Rest,Score,[V2|Phrase]) :-
   word(p,P),
   word(vnfc,V),
   word(ving,V),!,
   V2= V,
   fsa(T,Tail,Rest,Score,Phrase).


% fsa is looking for a verb of a certain kind.  if the next word is
% a verb, skip the current word if it is an auxiliary or a child of a auxp
fsa([H|T],[V1,V2|Tail],Rest,Score,[[V3]|Phrase]) :-
   member(H,[vpsed,ving]),
   word(vaux,V1),
   (word(v,V2)
   ;
   word(vpsed,V2)
   ;
   word(vaux,V2)
   ;
   (word(X,V2),
   child_of(X,advp))),!,
   V3= V1,
   fsa([H|T],[V2|Tail],Rest,Score,Phrase).




% verb, followed by adverb then accept the verb.
fsa([vpsed|T],[V1,Adv|Tail],Rest,Score,[V2|Phrase]) :-
   word(vpsed,V1),
   word(adv,Adv),!,
   V2= V1,
   fsa(T,[Adv|Tail],Rest,Score,Phrase).



fsa([vpsed|T],[C|Tail],Rest,Score,[C2|Phrase]) :-
   find_element(vpsed,C),
   !,
   C2= C,
   fsa(T,Tail,Rest,Score,Phrase).
```

```
% fsa is looking for a verb in plain, -s, or -ed form,
% but the current word is not it.  The current word is a part
% of a verb phrase so move past it.
fsa([vpsed|T],[V|Tail],Rest,Score,[[V2]|Tl]) :-
   word(Cat,V),
   (child_of(Cat,vp)
   ;
   child_of(Cat,auxp)),!,
   V2= V,
   fsa([vpsed|T],Tail,Rest,Score,Tl).









%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% noun special cases
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/*
% temporary rule.  take out if it doesn't work right
% stop if you have a pronoun followed by a verb
fsa([n|_],[Pr,Vb|Tail],Tail,0,[]) :-
   word(pronoun,Pr),
   word(v,Vb).
*/

fsa([n|_],[N1,N2|Tail],Tail,0,[]) :-
    N1 == that
    ;
    N2 == that.

% might add some rules here that
% look at the different types of words that follow nouns.

% if you have a word you think is a noun, if the
% next word is a verb and the noun is also a verb or aux verb then
% you are done parsing nouns
fsa([n|T],[n(N),V|Tail],Rest,Score,Phrase) :-
    (word(vpsed,N)
    ;
    word(vaux,N)),
    (word(vpsed,V)
    ;
    word(vaux,N)),!,
    fsa(T,[N,V|Tail],Rest,Score,Phrase).


% if you have a word you know is a noun followed
% by another that looks like it, label the possible as a noun
fsa([n|T],[n(N),N2|Tail],Rest,Score,[N1|Phrase]) :-
    word(n,N2),!,
    N1= N,
    fsa([n|T],[n(N2)|Tail],Rest,Score,Phrase).


% if you have a word you labeled as a noun,
% and you're looking for a noun, then accept the labeled one
fsa([n|T],[n(N)|Tail],Rest,Score,[N1|Phrase]) :-
```

```
    N1= N,!,
    fsa(T,Tail,Rest,Score,Phrase).


% if you have a candidate noun followed by a
% complementizer followed by a noun phrase, your first noun probably isn't one
fsa([n|_],[N,Comp,Det|T],T,0,[]) :-
   word(n,N),
   word(ving,N),
   word(comp,Comp),
   word(d,Det).




% if you havea pronoun followed by a
% verb, but are looking for two nouns, then quit.
fsa([n,n|_],[Pr,Vb|Tail],Tail,0,[]) :-
   word(pronoun,Pr),
   word(v,Vb).

% accept a pronoun as a noun if the next word is not part of a noun phrase.
fsa([n|T],[Pr,Vb|Tail],Rest,Score,[Pr2|Phrase]) :-
   word(pronoun,Pr),
   word(v,Vb),
   \+ word(n,Vb),!,
   Pr2= Pr,
   fsa(T,[Vb|Tail],Rest,Score,Phrase).

% if you see a plural noun followed by a
% singular noun, then accept the plural and move to detect the next element
fsa([n|T],[N1,N2|Tail],Rest,Score,[N3|Phrase]) :-
    word(npl,N1),
    word(nsg,N2),!,
    N3= N1,
    fsa(T,[N2|Tail],Rest,Score,Phrase).

   /*C1 \== pronoun,
   child_of(C1,dp),*/
   /*C2 \== d,
   C2 \== pronoun,
   \+ word(p,N2),
   child_of(C2,dp),*/


% fsa looking for a noun.  if the current word is a
% noun but not a pronoun, and
% the next word is also part of a noun phase, skip it.
% temp fix: we s
fsa([n|T],[N1,N2|Tail],Rest,Score,[[N3]|Phrase]) :-
   (word(n,N1)
   ;
   word(conj,N1)),
   (word(n,N2)
   ;
   word(conj,N2)),
   !,
   N3= N1,
   fsa([n|T],[N2|Tail],Rest,Score,Phrase).


% skip determiner and adjective if the adjective can also be a preposition
fsa([n|T],[D,A|Tail],Rest,Score,[[D1],[A1]|Phrase]) :-
   word(d,D),
   \+ word(n,D),
   word(p,A),
   word(adj,A),!,
   D1= D,
   A1= A,
   fsa(T,Tail,Rest,Score,Phrase).

% skip determiners and mark following nouns.
% this is necessary because some nouns are also verbs.
fsa([n|T],[D,N|Tail],Rest,Score,[[D1]|Phrase]) :-
    word(d,D),
    \+ word(n,D),
    word(n,N),
    D1= D,
    !,
    fsa([n|T],[n(N)|Tail],Rest,Score,Phrase).

% skip determiners
fsa([n|T],[D|Tail],Rest,Score,[[D1]|Phrase]) :-
   word(d,D),
   \+ word(n,D),
   D1= D,
   !,
   fsa([n|T],Tail,Rest,Score,Phrase).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% any category
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fsa([X|T],[C|Tail],Rest,Score,[C2|Phrase]) :-
   find_element(X,C),
   !,
   C2= C,
   fsa(T,Tail,Rest,Score,Phrase).



% fsa is looking for a verb in -ing form, but the
% current word is not it.  The current word is a
% part of a verb phrase so move past it.
fsa([ving|T],[V|Tail],Rest,Score,[[V2]|T1]) :-
   word(Cat,V),
   (child_of(Cat,vp)
   ;
   child_of(Cat,auxp)),!,
   V2= V,
   fsa([ving|T],Tail,Rest,Score,T1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% skip adjps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check two words out
fsa(Pattern,[Adj1,Adj2|Tail],Rest,Score,[[Adj3,Adj4]|Phrase]) :-
   word(adj,[Adj1,Adj2]),
   Adj3= Adj1,
   Adj4= Adj2,
   fsa(Pattern,Tail,Rest,Score,Phrase).


% the current word in the input stream is an adjective.
% Parse a chunk and move past it.
fsa(Pattern,[Adj|Tail],Rest,Score,[[Adj2]|Phrase]) :-
   (word(adj,Adj)
   ;
   (word(Cat,Adj),
   child_of(Cat,adjp))),!,
   Adj2= Adj,
   fsa(Pattern,Tail,Rest,Score,Phrase).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% process pps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The current word is a pp. it is followed by a
% determiner or a plural noun. skip the pp.
fsa(Pattern,[PP,D|Tail],Rest,Score,T) :-
   word(p,PP),
   (word(d,D)
   ;
   (PP \== to,
   word(n,D))
   ;
   (PP == to,
   \+ word(v,D))
   ;
   word(npl,D)
   ),
   parse(pp,[PP,D|Tail],Remain,Words,_),
   fsa(Pattern,[pp(Words)|Remain],Rest,Score,T).





%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rules for skipping advps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/*
% the current word in the input stream is an adverb.
% Parse a chunk and move past it.
% special case for verbs
fsa([vpsed|T],[V,Adv|Tail],Rest,Score,[V2,[Adv2]|Phrase]) :-
   word(v,V),
   (word(adv,Adv)
   ;
   (word(Cat,Adv),
   child_of(Cat,advp))),!,
   V2= V,
   Adv2= Adv,
   fsa(T,Tail,Rest,Score,Phrase).

*/

% 'but' is normally considered an adverb.  I can't tell
% whether it is or not. I have eliminated it from the lexicon.

% the current word in the input stream is an adverb.
% Parse a chunk and move past it.
% don't skip adverbs that are also subordinating conjunctions.
fsa(Pattern,[Adv|Tail],Rest,Score,[[Adv2]|Phrase]) :-
   \+ word(subconj,Adv),
   (word(adv,Adv)
   ;
   (word(Cat,Adv),
   child_of(Cat,advp))),!,
   Adv2= Adv,
   fsa(Pattern,Tail,Rest,Score,Phrase).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% unknown words
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% for now the fsa skips over unknown words.
% later we can make it more intelligent
```

```
fsa(Pattern,[UW|Tail],Rest,Score,[[UW2]|Phrase]) :-
   word(X,UW),
   X == unknown_word,
   UW2= UW,
   fsa(Pattern,Tail,Rest,Score,Phrase).



% the current word doesn't match the pattern
fsa(_,Rest,Rest,0,[]).

/*
find_element(vpl,Verb) :-
   word(vpl,Verb).


% find_element(+Element,+Target)
% succeeds if Element matches Target
find_element(vpsed,V) :-
   word(vpsed,V).


% find a verb in -ing form
find_element(ving,V) :-
   word(ving,V).
*/

% find a verb in any form
find_element(vpsed,V) :-
   word(v,V).

find_element(subconj,[W1,W2,W3]) :-
   ccl([W1,W2,W3],conjunction,List),
   member(subordinating,List).

find_element(subconj,[W1,W2]) :-
   ccl([W1,W2],conjunction,List),
   member(subordinating,List).

% find a subordinating conjunction
find_element(subconj,SC) :-
   ccl([SC],conjunction,List),
   member(subordinating,List).

find_element(n,Word) :-
   word(pronoun,Word).

% find a word of a particular category
find_element(Cat,Word) :-
   %Cat \== vpsed,
   word(Cat,Word).







child_of(a,b).






%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% miscelaneous predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% simply traverses all the rules in the
%grammar and asserts the children of particular heads
get_children :-
   rule(X,List),
   get_children_aux(X,List),
   fail,!.

get_children.

get_children_aux(_,[]) :- !.


% optional element
get_children_aux(X,[[H]|T]) :-
   \+ child_of(H,X),
   rule(H,List),!,
   assert(child_of(H,X)),
   get_children_aux(X,List),
   get_children_aux(X,T).

get_children_aux(X,[[H]|T]) :-
   \+ child_of(H,X),
   word(H,_),!,
   assert(child_of(H,X)),
   get_children_aux(X,T).



% next element is a manditory element
get_children_aux(X,[H|T]) :-
   \+ child_of(H,X),
   rule(H,List),
   assert(child_of(H,X)),
   get_children_aux(X,List),
   get_children_aux(X,T).

get_children_aux(X,[H|T]) :-
   \+ child_of(H,X),
   word(H,_),!,
   assert(child_of(H,X)),
   get_children_aux(X,T).



get_children_aux(X,[_|T]) :-
   get_children_aux(X,T).


convert_ccl_to_list :-
   ensure_loaded('ccl.pl'),
   tell('cclalt.pl'),
   forall(ccl(X,Y,Z),
    (tokenize(X,Result),
    write(ccl(Result,Y,Z)),
    write('.'),nl)),
   told.

% takes in an atom that may have spaces
% sin it and returns a list of the atoms in it
tokenize(Atom,Result) :-
   atom_chars(Atom,Chars),
   tokenize_aux(Chars,[],Result).

tokenize_aux([],List,End) :-
   atom_chars(Atom,List),
   End= [Atom].
```

```
tokenize_aux([' '|T],List,[Atom|Result]) :-
   !,
   atom_chars(Atom,List),
   tokenize_aux(T,[],Result).

tokenize_aux([H|T],Chars,Result) :-
   append(Chars,[H],CList),
   tokenize_aux(T,CList,Result).




% get subset of WordNet
get_wn_subset(OutFile,Number) :-
   ensure_loaded('c:\\active projects\\thesis\\prolog\\wn_s.pl'),
   findall((Num,Syn,Word,Cat,X),s(Num,Syn,Word,Cat,X,10),List),
   msort(List,ListS),
   reverse(ListS,ListR),
   tell(OutFile),
   print_list(ListR,Number),
   told.

print_list(_,0).
print_list([H|T],Num) :-
   write(s),write(H),write('.'),nl,
   Num2 is Num-1,
   print_list(T,Num2).
```