SYCORAX:

An Automated Analyzer of the Syntactic Complexity of English Text

by

William Cody Boisclair

(Under the direction of Michael Covington and Walter D. Potter)

Abstract

This dissertation describes the development and evaluation of a new analyzer for syntactic complexity known as SYCORAX: the **SY**ntactic **CO**mplexity **RA**ting e**X**pert.

SYCORAX, like several prior applications for automated analysis of syntactic complexity (e.g., Long et al., 2006; Channell, 2007; MacWhinney, 2011), is based on the Developmental Sentence Scoring (DSS) scale developed by Lee (1974). These existing applications, however, all have one significant limitation in common: they are all strictly based on immediate linear context within a sentence. It is evident from prior work (Channell, 2003; Judson, 2006) that certain syntactic structures involved in DSS are simply not apparent from linear context alone; indeed, many structures incorrectly analyzed by human raters are due to incorrect interpretation of local context (Lively, 1984).

In contrast, SYCORAX incorporates a newly-developed shallow dependency parser known as JED (**J**ust **E**nough **D**ependency) optimized for the dependencies which are important in DSS, and uses the resulting parse tree in the calculation of its DSS scores. Even without complete optimization of its DSS rules, the use of shallow parsing in SYCORAX produces a distinct overall boost in the accuracy of syntactic complexity scores on a variety

of manually-scored real-world transcripts, as measured using Pearson correlation coefficient and point-by-point accuracy, with no significant increase in execution time.

DSS has proven numerous times to be psycholinguistically valid. It was originally designed to identify language delays in children, and more recent experiments have found it to still be valid in that respect (e.g., Scarborough, 1990); in addition, it has been found to be of use in identifying language decline in adults (Cheung and Kemper, 1992; Kemper et al., 2003, 2004), distinguishing different forms of developmental delay (Finestack and Abbeduto, 2010), and even identifying Alzheimer's dementia (Kemper et al., 1993). It is believed that the improvement in its automated analysis by SYCORAX will prompt even further research regarding it, much as the prior project CPIDR (Brown et al., 2008) has done for semantic complexity (Covington et al., 2009; Jarrold et al., 2010; Engelman et al., 2010; Tsai, 2010).

INDEX WORDS: Natural language processing, Computational linguistics, Syntactic complexity, Developmental Sentence Scoring

SYCORAX:

An Automated Analyzer of the Syntactic Complexity of English Text

by

William Cody Boisclair

B.S., Mercer University, 2005

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2011

SYCORAX:

An Automated Analyzer of the Syntactic Complexity of English Text

by

William Cody Boisclair

Approved:

Major Professors:	Michael Covington Walter D. Potter
Committee:	I. Budak Arpinar Krzysztof J. Kochut

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2011

Acknowledgments

This dissertation would not have been possible without help and support from a number of individuals.

Perhaps the most significant influence was that of my advisors. All of them played an important part in steering me toward the focus of this dissertation, but Dr. Michael Covington in particular deserves special thanks. It was the CPIDR project, another graduate project that he had advised, that gave me the idea for SYCORAX.

Indeed, SYCORAX itself owes a great debt to everyone who has contributed to and used the CPIDR project over the past several years. Although the heart of the program is of course completely different, the user interface design of SYCORAX was closely based upon that of CPIDR, largely because of the praise that the program has received from researchers. A number of fixes incorporated into SYCORAX also came about because of flaws discovered by other researchers in CPIDR.

Yet it is those who are closest to me who truly deserve the most recognition.

I would like to thank my family for supporting me through my ever-increasing number of years in graduate school, and for giving me support in those moments when I felt overwhelmed or depressed.

I would like to thank the many friends I met at UGA, who helped provide a welcome distraction at the times I most needed it.

Finally, I must thank my girlfriend, Sarah Sturdivant, for being available when I most needed support, for giving me some useful suggestions as I developed and tested SYCORAX—and for simply being an awesome person all around.

Table of Contents

Chapter 1

Introduction

This dissertation describes the development and evaluation of a new analyzer for syntactic complexity, known as SYCORAX: the **SY**ntactic **CO**mplexity **RA**ting e**X**pert.

Like a number of prior applications for automated analysis of syntactic complexity (e.g., Long et al., 2006; Channell, 2007; MacWhinney, 2011), SYCORAX uses the Developmental Sentence Scoring (DSS) scale (Lee, 1974) to evaluate the syntactic complexity of input text. Further like these applications, SYCORAX not only outputs a final DSS score, but also shows the breakdown of the score into individual sentences and subscores, in a format similar to that shown in Lee's own examples.

Where SYCORAX radically differs from these existing applications is in its method of analysis. Although some of the existing programs incorporate approaches such as probabilistic inference or nondeterminism, the scoring algorithm at the heart of all of these programs is based on immediate context within a linear scan of the sentence. In contrast, SYCORAX incorporates a newly-developed shallow dependency parser, and performs its DSS analysis based on links within the dependency trees generated by this parser. As will be discussed further, there are a number of syntactic structures involved in DSS which simply are not apparent from linear context alone; it is on these structures that SYCORAX demonstrates an advantage.

The inspiration and the development process of SYCORAX are described in further detail through Chapter 4. The process of evaluating SYCORAX in comparison to existing tools, as well as of debugging SYCORAX to further improve its accuracy, are then described in Chapters 5–6; here, it is shown that SYCORAX does indeed offer an improvement on the state of the art in automated DSS analysis due to the incorporation of parsing. Finally, known issues with SYCORAX and future development plans are laid out in Chapter 7.

1.1 Why Syntactic Complexity?

Before describing the development of SYCORAX in further detail, however, it is first necessary to give some background on how syntactic complexity is defined, why it is useful beyond mere theory, and why an automated analyzer for it is so badly needed.

Syntactic complexity is a general term used to describe a variety of quantitative measurements regarding the grammatical structure of a sentence. Measures of syntactic complexity can identify how difficult any given sentence is to comprehend, as well as how hard a sentence is to produce; in both of these cases, higher complexity indicates greater difficulty.

Because neurological and psychological disorders can affect the acquisition and production of language in an individual, syntactic complexity can be used for diagnostic purposes. Entire scales of complexity have been developed with the purpose of identifying developmental delays in children through samples of their language use (Lee, 1974; Scarborough, 1990); others have been created to measure the linguistic competence of developmentally disabled adults (Rosenberg and Abbeduto, 1987). Even in those with normal language development, it is known that linguistic complexity tends to decline with age, as discussed in the literature review of Cheung and Kemper (1992); this is further exacerbated by conditions such as Alzheimer's. Indeed, research has found several of these scales to be equally applicable to identifying language decline in aging adults (Cheung and Kemper, 1992; Kemper et al., 2003, 2004) and to diagnosing Alzheimer's (Kemper et al., 1993; Lyons et al., 1994; Snowdon et al., 1996, 2000).

It is obvious, then, that these metrics are relevant from a psychological perspective. However, they are not only tedious to calculate by hand, but are also prone to human error. This has long been a known problem: Lively (1984) identified a list of frequent errors made by raters using Lee's DSS scale, some of which can significantly affect a complexity score. Furthermore, when dealing with ambiguous sentences, there is room for interpretation on the rater's part; it is not uncommon for raters to interpret and thus score the same sentence differently. Indeed, Lee (1974) includes a real-world example of this: in two separate transcripts (Charts 15 and 19 from Lee), the sentence *They fall* is scored differently by two different raters, one of whom interprets the verb as being in the wrong tense.

An automated tool to analyze syntactic complexity would solve all of these problems. It would eliminate the tedium of manually scoring an entire corpus of utterances, and allow for more analyses to be performed in a shorter amount of time. It would be reliable, unlike human raters, producing the same result on the same sentence every time. Ideally, if its rules were written as accurately as possible, it would also be less error-prone than humans.

1.2 CPIDR: The Inspiration for SYCORAX

The project which most strongly inspired SYCORAX was CPIDR (Computerized Propositional Idea Density Rater), an application developed in 2007 at the University of Georgia (Brown et al., 2008). Like SYCORAX, it is an automated utility to measure the complexity of utterances; however, unlike SYCORAX, what it measures is their *semantic* complexity.

Semantic complexity is a measure of the number of ideas expressed within a text; in short, a more semantically complex text expresses a greater variety of meaning. The most popular measure of this within psycholinguistic literature is *idea density*, defined in Kintsch and Keenan (1973) as a ratio of the number of propositions expressed in a sentence over the number of words in the same sentence; this is what CPIDR measures.

On a novel corpus of 80 transcripts, CPIDR correlated extremely well with human analyses ($r = 0.97$), significantly better even than among five human raters on a subset of the same transcripts ($r = 0.81$). It has since been used in research on such varied topics as schizophrenia (Covington et al., 2009), Alzheimer's disease and aging in general (Jarrold et al., 2010; Engelman et al., 2010), and teaching of English as a second language (Tsai, 2010).

However, as Cheung and Kemper (1992) have shown, semantic and syntactic complexity do not directly correlate with one another. A sentence may express a wide variety of ideas using syntactically shallow structures such as multiple modifiers; it may also use syntactically complex structures that carry little meaning. In addition, semantic and syntactic complexity correlate differently with other non-linguistic factors. All but one measure of syntactic complexity tested by Cheung and Kemper (1992) correlated negatively with age of adults, but no such correlation existed for semantic complexity. In the opposite direction, the Nun Study of Snowdon et al. (1996, 2000) found a correlation between reduced idea density and later development of Alzheimer's disease in samples written decades before the subjects showed evidence of Alzheimer's, but no such correlation for syntactic complexity. This was later corroborated by Engelman et al. (2010) using a vastly different sample population, this time of medical students at Johns Hopkins University.

Clearly, there is a need for a syntactic complement to CPIDR—and it was this realization that led to the development of SYCORAX.

1.3 Choosing a Measurement

The first problem in developing an automated syntactic complexity analyzer is that, unlike semantic complexity, no measure is universally agreed upon. There is also no easy way to derive any one measure from another; there are notable differences in the specific structures analyzed, the level of detail in which they are analyzed, and the extent to which a score can be broken down into sub-scores for further analysis.

Perhaps the best comparison of the variety of approaches to syntactic complexity can be found in Cheung and Kemper (1992), which compares several popular measures of complexity as applied to the language use of adult English speakers over several decades. Aside from the single measure of semantic complexity (idea density, as discussed above), these measures can be divided into three main categories:

1. **Length-based:** Mean length of utterance (MLU) and mean clauses per utterance (MCU). These are the most naïve analyses, as they are only concerned with the number of words or clauses in a sentence, with no attention paid to how those words or clauses are actually combined.

2. **Comprehension-based:** Yngve depth (Yngve, 1960) and Frazier count (Frazier, 1985). These are based on the depth of embedding in a phrase-structure tree, and were designed to predict the level of difficulty that a listener or reader would have in processing a sentence.

3. **Production-based:** Developmental Sentence Scoring (DSS; Lee, 1974), Index of Productive Syntax (IPSyn; Scarborough, 1990), Developmental Level (D-Level; Rosenberg and Abbeduto, 1987), and Directional Complexity (Botel and Granowsky, 1972). All of these are based on the appearance of certain categories of syntactic structure—e.g., subordinate clauses, objects of verbs, and conjunctions. These structures are scored according to their ordering in language acquisition: those which typically manifest at a younger age are ranked lower than those which manifest at a higher age. This allows novel utterances to be ranked based on the order they might appear in language development.

Most of the body of literature using syntactic complexity as a diagnostic metric focuses on the production of novel sentences, not the comprehension of existing ones; even studies which incorporate comprehension-derived complexity metrics, such as that of Cheung and Kemper (1992), still use those metrics to measure language production. The studies of semantic complexity incorporating CPIDR, as discussed in Section 1.2, have also analyzed newly produced utterances. Furthermore, as Voss (2005) observes, metrics based on language production tend to be easier to analyze computationally than those based on comprehension; the latter typically require a full parse of the sentence, while the former only require partial parsing at most.

The combination of all of these factors suggests that a production-based approach to syntactic complexity would be preferable for SYCORAX. Out of this class of metrics, the most frequently cited in psycholinguistic literature have been DSS (Lee, 1974), D-Level (Rosenberg and Abbeduto, 1987), and IPSyn (Scarborough, 1990).

IPSyn is quite detailed in its linguistic profile; indeed, for the population for which it was designed, it seemed quite promising. Holdgrafer (1995) found that IPSyn was more useful than DSS in distinguishing neurologically typical preschoolers from language-delayed preschoolers between the ages of 3 and 5, and Rescorla et al. (2000) shows further evidence that IPSyn is a useful indicator of language impairment prior to age 4. Scarborough, however, commented in his own 1990 paper that IPSyn had not been tested beyond age 4, and that validity beyond that age group would require further study.

Unfortunately, later research on IPSyn revealed that Scarborough's concerns were very valid. Cheung and Kemper (1992) found IPSyn to correlate better than any other measure of syntactic complexity with education and vocabulary in a population of senior adults, but *worse* than any other with the more psychologically relevant factors of age and digit span. Rescorla et al. (2000) found that though a correlation existed between MLU and IPSyn in language-disordered children of age 4, no such correlation was present for neurologically typical children of the same age. Hewitt et al. (2005) found IPSyn to be less useful than even MLU for identifying language impairment in a population of average age 6; the latter identified 67% of language-impaired children, while the former only identified 37%. Minch (2009), using a computerized analysis of a corpus derived from the speech of three elementary school grades and a college class, found that IPSyn correlated only weakly with all other syntactic measures studied, including syntactic productivity as measured by the number of unique structures occurring with a certain frequency.

It had become clear that IPSyn was only valid within a limited age range, and significantly less relevant outside that range; that, then, left D-Level and DSS. D-Level was developed for the analysis of language use in adults with mental retardation (Lee, 1974), but was also famously used in the Nun Study of Alzheimer's (Snowdon et al., 1996, 2000). DSS, on the other hand, was developed with an eye toward childhood language usage, and a large majority of the research using it has naturally focused on children (e.g., Mayberry, 1973; Politzer, 1974; Pierce and Bartolucci, 1977). However, like D-Level, it has been used in studies of adults with Alzheimer's (Kemper et al., 1993; Lyons et al., 1994); in addition, it has been used to analyze the language of adults with other conditions that are known to affect language acquisition, such as Down syndrome (Kernan and Sabsay, 1996) and deaf-blindness (Chomsky, 1986). Both D-Level and DSS have also been used to analyze the change in the complexity of sentences produced by aging adults, and both have been found psychologically valid for that purpose (Cheung and Kemper, 1992; Kemper et al., 2003, 2004); further validation of DSS, but not D-Level, was provided by Minch (2009).

Further comparison of these two scales revealed that D-Level, though simpler, has several downsides to its simplicity. Perhaps the most notable is that, because of the method by which D-Level is calculated, the presence of more complex structures overshadows the presence or absence of less complex structures. Yet those lower-ranked structures are still significant; a speech sample incorporating higher-level structures without the use of lower-level structures may still be an anomaly. Worse, a combination of any two or more of the structures from levels 1–6 is scored as 7, regardless of which forms were combined. Rosenberg and Abbeduto themselves observed that the vast majority of the sentences in their test corpus scored at level 7, thus requiring further analysis for the results to be truly meaningful. The accuracy of the D-Level scale's developmental sequence has also been brought into question by Covington et al. (2006).

DSS also produces a single value as its ultimate result, but arrives at it via a different method. Rather than compressing all of syntax into a single scale, DSS is instead calculated as the sum of scores from eight distinct scales, each of which pertains to a particular category of syntactic structure. Because of this, DSS analyzes a broader selection of structures than D-Level; while all of D-Level's scores relate to the combination of simple clauses into a more complex sentence, DSS also analyzes such features as negation, pronoun types, interrogatives, and question inversion, all of which contribute to the complexity of syntax. In addition, each scale in DSS is cumulative: if someone uses multiple structures from a category, the score for that scale is the sum of the scores of all relevant structures. Yet much like D-Level, DSS has also been criticized regarding the accuracy of its developmental scale; this will be discussed in further detail in Chapter 2.

In the end, I chose to focus specifically on DSS for a number of reasons. It was more widely used in studies of both children and adults; it identified a wider variety of structures than D-Level; unlike D-Level, its scores were cumulative; it could be broken down into component scores, allowing syntactic anomalies to be spotted more easily; and as shown by Cheung and Kemper (1992), it correlated reasonably well with D-Level.

1.4 Existing DSS Analyzers

Before developing a new automated analyzer for DSS, it was necessary to review the few existing applications for automated DSS scoring to ensure that it would indeed be possible to contribute something new to the field.

Attempts at automated Developmental Sentence Scoring date back to 1983, when Peter Hixson developed the aptly-named Computerized DSS for the Apple II (reviewed in Klee and Sahlie, 1986). As is to be expected of a program released in that year, this application was extremely primitive by modern standards; although it did ease the process of performing a DSS analysis, it could hardly be considered fully automated. Most notably, Hixson's application required a great deal of additional manual coding to be applied to a transcript to produce any sort of meaningful result. Irregular verbs and present-tense plural verbs were not automatically identified and had to be explicitly marked with symbols (`<` and `V`, respectively). Incorrect conjugations also had to be indicated using a special notation; for instance, a use of *is* when *are* was correct would have to be written as `**is*are`. Even with these notational quirks taken into account, certain pronouns and auxiliary verbs were still not scored at all, while others were given incorrect scores.

The next major tool to automatically analyze DSS was CLAN (Computerized Language ANalysis), a multi-purpose tool originally developed in 1991 as part of the CHILDES project (MacWhinney, 2000). In 2004, it received a significant update which added automated part-of-speech tagging, potentially making it a viable automated DSS analyzer. However, its relative accuracy is still unknown; the relevant improvements to the program are recent, and no studies of the program have been performed since then. Channell (2003) was unable to find any literature estimating the accuracy of CLAN on DSS when the tagging improvements were still unimplemented, and the same still appears to be true even after its implementation. This dissertation thus incorporates one of the first thorough tests of CLAN's DSS accuracy, the results of which are discussed in Chapter 6.

The 2004 version of CLAN uses a three-step approach to analyzing DSS: the sentence is analyzed morphologically to find all possible tags for each word, the morphological analysis is disambiguated by context, and the DSS score is then calculated from the resulting tagged text. Even in more recent versions, however, CLAN still cannot identify certain structures automatically. One sentence which MacWhinney gives as an example is *What this say?*; this sentence should receive attempt marks for both primary verb and interrogative reversal, but neither error is identified automatically. MacWhinney identifies three additional forms which are entirely ignored by his automated analyzer: the word *one* used as a pronoun, the distinction between complementing and adjunct infinitives, and embedded clauses without subordinating conjunctions (e.g., *the man* ***we saw yesterday***).

Another application which can perform automated DSS is Computerized Profiling (CP), originally developed by Steven H. Long in 1986 and extended in the following years by Ron Channell, with the latest update to its DSS scoring routines having been added in 2000 (Long and Channell, 2001; Channell, 2003). Unlike CLAN, the accuracy of Computerized Profiling has been extensively tested, as described in Channell (2003). Although a high correlation

(r = 0.97) did exist between manual and automated per-sentence scores, the automated scores were on average three-fourths of a point higher; worse yet, point-by-point agreement (i.e., agreement of individual sub-scores per sentence) between CP and manual scores was a surprisingly low 78.2%, a result which is clearly problematic for anyone who needs to look beyond the final DSS score. A number of less common structures were not identified at all by CP's automated analysis, including participles, passive infinitives, and inversion of verbs with auxiliary *have* or multiple auxiliaries.

DSSA, a later program by Channell, scored somewhat better on a different set of transcripts, with 86% point-by-point agreement and $r = 0.98$ (Judson, 2006). It performed more accurately on all of the forms which proved troublesome for CP, with the exception of interrogative reversals. However, the point-by-point agreement of DSSA is still well below the 96% average agreement among trained speech-language pathologists on the same set of transcripts. Furthermore, DSSA is still new enough that the only reference found in a search of relevant literature was that of Judson. Although the latest version was compiled in 2007 (Channell, 2007), the program was only released to the public by Channell in 2011; prior to this year, only Channell's own advisees had access to DSSA.

Although the existing applications for DSS which have been tested produce high correlations with manual analyses for overall DSS scores, it is obvious, given point-by-point agreement, that the individual sub-scores are significantly less accurate—a definite problem if one needs to analyze patterns involving specific structures. Long and Channell (2001) cite a threshold of 85% accuracy as a measure of acceptability; by this standard, the 78% agreement of CP is unacceptable, and the 86% agreement of DSSA is only barely acceptable. A good analysis is said to be 90% or better, and an excellent analysis 95% or better; ideally, it would even be possible to reach the inter-rater agreement of 96% observed for human raters.

Worse yet, both CLAN and DSSA have a rather significant flaw: neither of them attempts to analyze what Lee terms the *sentence point*, a point added to the DSS score for any grammatical sentence. Unlike Channell's study of CP, Judson's analysis of DSSA completely ignored the sentence point when calculating point-by-point agreement, producing a skewed accuracy result as concerns overall scores. It is true that the sentence point is largely negligible in the correlation coefficient, as it is only a single point difference between scores; however, it is significant with respect to point-by-point agreement, as it has the same weight as every other mark. It has also been shown to be a significant factor in specific studies incorporating DSS; for instance, Finestack and Abbeduto (2010) found the sentence point to be useful in distinguishing between two types of developmental delay.

The aim of this project, therefore, is to produce a new completely automated DSS rater which will exceed the accuracy of the existing automated solutions for DSS. As discussed above, Channell has provided two useful metrics by which to measure DSS accuracy: point-by-point agreement and the Pearson correlation coefficient. As the latest versions of CLAN, CP and DSSA are all offered as freeware, they can be run on the same language samples as SYCORAX for the purpose of comparison. Moreover, the public release of DSSA includes a subset of the corpus used by Judson (2006), thus allowing for a comparison among the applications on a reasonably large sample of real-world data.

1.4.1 A Need for Further Analysis

As mentioned in the introduction to this chapter, a common factor amongst all existing DSS analyzers is that they use a linear, surface-level analysis of text in order to score it. Some of them have introduced modifications to improve the analysis; for instance, CP performs a LARSP (Language Assessment, Remediation and Screening Procedure; Crystal et al., 1989) analysis on the sentence before calculating its DSS score, while DSSA uses probabilities derived from actual language samples to better disambiguate the context of words. However, even with these enhancements, the existing applications are still limited to local context in making certain distinctions—and not all distinctions necessary to DSS can be made using such a surface-level approach.

Rules Needing Parsing

For instance, none of the above-mentioned applications can fully test for subject-verb agreement, despite the fact that verbs must agree with their subjects in order to earn anything more than an attempt mark in DSS. The main problem, in this case, is that subject-verb agreement is *not* merely a matter of searching for the nearest noun or pronoun.

For relatively simple sentences such as *The dog barks*, using strictly local context is perfectly appropriate. For more complex sentences, however, heuristics based on local context may not apply. Consider, for instance, the sentence *The dogs in the kennel are barking.* Here, the actual subject of *are barking* is *dogs*; *kennel*, the noun that is closest, is actually the object of a preposition that in turn modifies *dogs*. In order to properly analyze the subject-verb agreement in this sentence, and thus give an appropriate score, it is necessary for the analyzer to recognize that *are barking* should agree with *dogs*, not *kennel*.

Subject-verb agreement can affect not only the main verb score, but also the score for negatives. The early occurring forms *don't* and *isn't* are only scored if they are used correctly; in other words, their subjects must agree with those particular verb forms, and not *doesn't* or *aren't*. This, too, requires identifying the subject to ensure agreement, in this case with the auxiliary verb that is negated; otherwise, the sub-scores for both main verbs and negatives will disagree with a manually calculated score.

This analysis becomes even more difficult when conjunctions are involved, particularly when the conjunction is one of verbs. When two verbs are joined by a conjunction, each is to be scored just as if it stood separately, with the final verb score being their sum. However, when the first verb is a compound verb, the second verb is obligated to delete any auxiliaries. For instance, consider the sentence *They were eating chicken and drinking tea*; here, *were* is an auxiliary of both *eating* and *drinking*. There is no easy way to identify that both verbs have an auxiliary using only a simple linear scan of the sentence, particularly with the intervening direct object. Furthermore, subject-verb agreement must apply to all of the verbs joined by a conjunction; for instance, **He eats potato chips and drink soda* is incorrect, and in this case, the first verb should be given a full score while the improperly conjugated second verb should only receive an attempt mark.

On the topic of deletions, there is also the fact that certain verbs require accompanying infinitives to be *bare*—that is, to delete the marker *to*. In order to properly analyze whether these structures are grammatically correct or only deserving of an attempt mark, it is necessary to determine which verb is the main verb, whether the second verb depends on the main verb, and whether the main verb requires infinitives to be bare. The problem is that intervening words may change the function of the secondary verb; compare, for instance, *I see the children eat* (in which *eat* is a bare infinitive) and *I see what the children eat* (in which *eat* is a simple uninflected verb in a subordinate clause). Any attempt to analyze every possible sentence structure in which an apparent bare infinitive was actually part of a subordinate clause would become awkwardly complex.

These are not merely theoretical concerns, as can be shown using the existing DSS applications. Table 1.1 shows the errors made by the three leading automated DSS analyzers on all of the structures discussed above. Interestingly, the form of the error can differ significantly among the three applications; for instance, in the *eating and drinking* sentence, CP interprets *eating and drinking* as a compound participle, DSSA interprets *were eating* as a main verb but *drinking* as a participle, and CLAN ignores *drinking* entirely. Nonetheless, each of these structures produces an error in at least one of the three automated DSS applications, and most produce scoring errors in all three.

Rules Aided By Parsing

In addition to the above, there are other rules for which a linear heuristic-based approach could greatly be simplified with the addition of parsing. One such rule is the case of interrogative reversals and missing auxiliary verbs: namely, how does one determine that there is not just a missing auxiliary, but also that an inversion was not made where it should have been? Examples of questions with both missing auxiliaries and inversions can be found in

Table 1.1 **Errors made by the three leading DSS analyzers on error-prone sentences. The bolded word represents where the error would occur; an 'X' indicates that an error was made in analyzing that word.**

Sentence	**CP**	**CLAN**	**DSSA**
The dogs in the kennel **are** barking.		X	
*The dogs in the kennel **is** barking.	X	X	X
The dogs in the kennel **don't** bark.		X	
*The dogs in the kennel **doesn't** bark.	X	X	X
They were eating chicken and **drinking** tea.	X	X	X
I see the children **eat**.	X	X	X
I see what the children **eat**.			

sentences 27 (*What you eating?*) and 30 (*You want to get spanked?*) of Chart 10 from Lee (1974), as reproduced in Appendix C of this dissertation. The solution seems simple at first: In nearly all cases, inversion is necessary in questions. An auxiliary is also obligatory in these cases, unless the verb is the copula. Indeed, a linear scan of the sentence would be sufficient for most cases: a sentence is a valid question if an auxiliary precedes a noun phrase which then precedes a main verb.

There is one significant exception to this rule, however, and it concerns interrogatives. As discussed in section 3.2.1.2 of Huddleston et al. (2002), when the subject is an interrogative phrase, it is obligatory not to invert (e.g., *Who wrote this program?*, not **Did who write this program?*); however, when the fronted object is an interrogative phrase, inversion is still necessary (e.g., *What does this program do?*). A purely context-based approach to this problem would quickly become unwieldy; the ability to distinguish subjects from objects would make this analysis much easier.

Other examples of distinctions that are made easier to analyze with the help of a parser—and likely more accurate as well, as a side effect—include gerunds and participles, double negatives, and pronoun case. The functions of gerunds and participles are made completely distinct by a parser, with the former acting as a complement and the latter acting as an adjunct, and no awkward heuristics would be necessary in the DSS analyzer to distinguish them. Nesting of clauses can be observed in the case of double negatives, to ensure that only a single negative at most applies to each clause. Pronoun case is trivial to analyze if subjects and objects can be distinguished by the parser.

1.5 A Hypothesis

Clearly, given the above discussion, and especially given the results shown in Table 1.1, it would be beneficial to have some sort of parser integrated into SYCORAX, so that it would be less likely than existing applications to make certain types of errors that are obvious to any human rater sufficiently trained in DSS.

At the same time, however, it seems that a full parse is unnecessary. For the purposes of DSS, some structures are more important than others; for just one example, it is necessary to distinguish subjects from objects, but not to distinguish direct from indirect objects. A comprehensive parser would be overkill for the purposes of SYCORAX, as it would take too much time to analyze structures that are entirely unnecessary for DSS analysis.

Thus, the hypothesis of this dissertation is that the addition of a domain-specific shallow parser, combined with a set of appropriately detailed DSS rules, will make SYCORAX's automated scores more accurate than even DSSA on a variety of texts, as measured by both point-by-point agreement and correlation coefficient. As the existing applications are available as freeware, and as a variety of manually-scored transcripts exist in Lee (1974), Lively (1984) and the files included with Channell (2007), this hypothesis is readily testable once SYCORAX has been sufficiently developed.

Chapter 2

Modifying the DSS Scale

The Developmental Sentence Scoring scale defined in Lee (1974) has continued to be used in psycholinguistic research decades after it was originally developed. Hughes et al. (1992) made the case that it had "aged rather gracefully," in spite of the outdatedness of the syntactic and psycholinguistic theories on which it was based. That same year, Cheung and Kemper (1992) showed that DSS accurately identifies the decline of syntactic complexity in senior adults; a year later, Kemper et al. (1993) found a similar correlation with a standard measure of Alzheimer's dementia, largely due to a decrease in conjunction and subordination.

Well over a decade after Hughes et al. (1992), researchers still continue to incorporate DSS into research on language development. Kemper et al. (2003, 2004) found different patterns of complexity in the sentences produced by younger and older adults given the same prompts. Minch (2009) found that DSS was more valid than the much newer Index of Productive Syntax (Scarborough, 1990) for elementary-school children and college-age adults alike. Finestack and Abbeduto (2010) found that a breakdown of DSS results can distinguish the language patterns of typically developing adults, adults with Down syndrome, and adults with Fragile X syndrome from one another.

Nonetheless, much like Rosenberg and Abbeduto's D-Level, for which a revision has been presented by Covington et al. (2006), DSS has also faced a number of criticisms, with various modifications suggested to address these concerns.

2.1 Criticisms of DSS

One criticism of DSS parallels Covington et al.'s criticism of D-Level: that its scoring rubric does not accurately reflect the order in which syntactic features appear in childhood language development as was originally intended. This was already evident just slightly over a decade after Lee's book was published; Klee and Sahlie (1986) observed that research on language acquisition published in the intervening years had rendered some of the claims about language development which influenced Lee's scoring system to be invalid. Indeed, this may even be worse for DSS than for D-Level, as the former predates the latter by thirteen years—a significant length of time as far as theories of language acquisition are concerned.

Note that this is specifically a problem of psycholinguistic theory, and *not* of formal syntactic theory. It is indeed true that syntactic theory has evolved significantly since Lee published her book. However, as concerns language development, this point is moot: in DSS, syntactic theory is only used as a means to an end, as a way to formalize empirical observations regarding children's language. The order of development is the same, regardless of how it is analyzed; indeed, as will be further discussed later in this dissertation, DSS can be analyzed by means other than the phrase-structure parsing that was originally used to define it. In addition, this does not mean that DSS has been invalidated as psycholinguistically useful; as discussed above, research within the past decade has shown DSS to still be psychologically relevant across a variety of age groups.

Another criticism concerns the fact that, according to the guidelines originally proposed by Lee, DSS scores are limited to the first fifty complete sentences in a block of text. The reasons behind this decision are understandable—not only does it decrease the amount of time needed to rate a sample of text, it also ensures that, for the sake of normalization, all samples being compared are of equal length. Yet it is not uncommon for a text to vary in the complexity of its sentences; as discussed by Johnson and Tomblin (1975), using a 50-sentence segment may produce results which are misleading for certain samples of speech or writing, even when the segment has been selected at random. The time constraint is negligible for computerized analysis; one of the reasons that automated analysis of DSS is so appealing is that it is significantly faster than the same analysis done by hand.

Third, DSS suffers from another criticism which applies to D-Level: what is, in this case, quite a detailed analysis of syntactic patterns is condensed into a single monolithic score that can be somewhat misleading. Two distinct patterns of language use may earn the same final score, as Hughes et al. (1992) observe; indeed, as Hughes points out, even Lee herself urged researchers to look beyond the final score and into specific details. Research such as that of Kemper et al. (2003, 2004) and Finestack and Abbeduto (2010) has further confirmed that some patterns are only observable by looking at individual sub-scores in addition to the overall score.

2.2 Improving DSS

Despite there being room for improvement in DSS, the original unmodified version of the scale, as summarized in Appendix B, *must* be kept intact in SYCORAX as an option. Any modifications, as Hughes et al. (1992) emphasize, would render comparisons with scores based on the original rules invalid, including the age norms described by Lee. Thus, it is important that any modifications to DSS can be turned off; this choice can easily be implemented in SYCORAX through check boxes in the user interface, with conditional statements in the analysis itself enabled only when the relevant boxes are checked.

The simplest improvement would be an option to allow the score to be calculated using the full transcript rather than a fifty-sentence sample. This is trivial to implement, simply by adding a conditional statement within the analysis loop that breaks out of the loop after the fiftieth sentence if the option remains unchecked.

The second simplest improvement is to provide more information than a single final score. This problem had already been solved by prior applications for automated DSS analysis; CP, CLAN and DSSA all output the individual category scores for each sentence in addition to the overall score. In the case of SYCORAX, I chose to use a tab-delimited format, with each tab stop representing one of the categories of DSS; within each category, individual scores are then separated by commas. This format has the advantage of being both easily read by humans, as it closely resembles the format of the tables used in Lee (1974), and easily processed by a computer, as it is based entirely on two types of delimiters. An example of this format for Chart 10 of Lee (1974), reproduced in its original format in Appendix C, is shown in Figure 2.1.

	Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Point	Total
1			-						0	0
2			-						0	0
3			1						1	2
4			2						1	3
5		3,3	2						1	9
6		3,3	6		7				1	20
7		3,3	6		7		6		1	26
8		3,3	6		7		6	7	1	33
9		3	inc.		7		6	7	1	24
10	3		7						1	11
11	3	1	1	2					1	8
12	3	1,2	1	5					1	13
13	3	1	2	5					1	12
14		1,3	6				6		1	17
15		1,3	6		7		6		1	24
16	4	3	2	5		8			1	23
17		2,3	2,2			8		2	1	20
18		1,1,3	2,6		7	5			1	26
19		1	1	8					1	11
20	4	1,1,3	2,6		7	8			1	33
21	7	1	2						1	11
22	3,3		8						1	15
23	3,3		1	2					1	10
24		3	2,4		4				1	14
25	1	6,2	1,2		5				1	18
26	-	-	4		4				0	8
27		1	-				-	2	0	3
28	7	-	-		-				0	7
29			-						0	0
30		1	-	7			-		0	8
Total	47	73	93	34	62	29	30	18	23	409
Avg	1.5667	2.4333	3.1000	1.1333	2.0667	0.9667	1.0000	0.6000	0.7667	13.6333

Figure 2.1 **The correct DSS analysis from Chart 10 of Lee (1974) shown in SYCORAX's output format.**

That, then, leaves revising the scoring system itself. Unfortunately, in this case, the solution is not so clearly defined. Although Klee and Sahlie (1986) criticize DSS on the basis of its outdated developmental model, they do not cite even one example of how newer research has rendered DSS invalid, and instead leave this as an exercise for later researchers.

Despite over three decades having passed since the publication of DSS and two decades since Klee and Sahlie's review, however, I was unable to find any more recent literature outlining revisions of Lee's scale to the extent suggested by Klee and Sahlie. Such a thorough revision would require reviewing a large mass of literature written since the early 1970s on theories of language acquisition, determining what is applicable to DSS and how its rules would be affected, developing appropriate modifications to those rules, and ideally, testing the updated scale to determine the validity of these modifications.

Clearly, reviewing and revising DSS to this extent is beyond the scope of this dissertation; improving the accuracy of automated DSS using the unmodified scale is enough of a challenge in itself. However, though no comprehensive revision of DSS could be found, several papers did suggest smaller-scale improvements to DSS rules which would bring DSS more in line with modern theories of language acquisition.

One such paper was the aforementioned work of Hughes et al. (1992). In that paper, Hughes et al. suggest nine modifications to the DSS rules, two of which are significant divergences from Lee's rules and the other seven of which are clarifications of existing rules. These can be summarized as follows, with the first two rules being the significant changes:

1. *Like* is not scored when it is used as a preposition, only when it is used as a subordinating conjunction.
2. All sentences containing a subject and verb in the same clause are included. The entire sentence will be scored, except for conjunctions that stand alone, that start a sentence, or that are preceded only by minor sentences.
3. *Then* is ignored as a conjunction, but a sentence containing it still earns a sentence point. *And then* is still counted as a conjunction.
4. An attempt mark is not given for interrogative reversal if the question is a request for clarification.

5. Incomplete utterances are given a sentence point if they contain a subject and verb, if the incompleteness is the result of interruption. The sentence point is deducted if there was no interruption.
6. All repetitions, even if they differ from the prior sentence by the addition or deletion of a minor element, are ignored in analysis.
7. Imitations of another speaker are ignored, even if they differ from the sentence being imitated by a minor element.
8. Present-tense *be* with *–en* verb is counted as a copula and adjective, unless the adjective is modified by a prepositional phrase. Past-tense *be* with *–en* verb may be considered passive if the verb does not change meaning by adding a prepositional phrase.
9. *Be supposed* is scored as a regular *be* verb, even if *supposed* is conjugated incorrectly. The infinitive following *supposed* is scored normally (i.e., 5 points).

Notably, some of Hughes et al.'s clarifications, much like some of the original rules in DSS, depend strictly on context, and there is no way that an automated program will be able to identify all of these contexts. For instance, Rule 4, the rule for question inversion, depends on the previous statement by the conversational partner, which may not even be included in the input transcript; some other notation may therefore be necessary to identify these sentences. Similarly, interruption could be indicated using a standard notation, such as a dash, but there is no guarantee that all transcriptionists will use the same conventions. There is also no easy way for a computer to identify, in the case of rule 8, that "the money was lost" does not change meanings with an added prepositional phrase but "the girl was lost" does—and even if the program did account for that distinction by means of a lexicon, it may not know how to deal with an unfamiliar word, as in "the Athenian was lost."

However, apart from these few context-sensitive rules, the analysis will usually be fairly straightforward even with these modifications in place. These context-sensitive cases are also rare, and there is usually one interpretation that is more prevalent: for instance, with question inversion, un-inverted questions are more likely to occur than requests for clarification, and the latter will not occur at all in transcripts of monologues.

DeThorne et al. (2008) not only implemented the changes from Hughes et al. in their own experiments involving DSS, but also introduced another modification regarding the scoring of initial conjunctions. Lee's rubric errs on the side of caution and ignores all conjunctions at the start of an utterance, even though Lee admits that some are syntactically significant. However, Lee also marks initial conjunctions in the transcript that are to be ignored using the standard convention of parentheses; thus, any *unmarked* initial conjunction is likely significant and should be scored.

One additional modification was made by Kemper et al. (1993): rather than counting the verbs of relative and subordinate clauses in the main verb category, as was done by Lee, Kemper et al. instead counted them as the highest two scores in the secondary verb category in their own experiment incorporating DSS. However, they cite no literature supporting this decision—most notably, there is no evidence given to show why these embeddings were ranked at the same levels as passive infinitives and gerunds, respectively. Even more problematically, this change breaks a significant developmental division that already existed in Lee's original version of DSS. All other verb forms counted in the Secondary Verbs category *cannot* be used as main verbs, and thus must be specifically learned in the context of secondary verbs. Subordinate clauses, on the other hand, contain an entire sentence as a constituent; thus, although they are secondary verbs in the context of the parse tree as a whole, subordinate verbs are in the same class as main verbs from a language-acquisition perspective. Because of Kemper et al.'s lack of justification or explanation for this particular modification, I ignored this change in my own modifications to DSS.

A couple of other modifications that I introduced in SYCORAX's modified rule set had no prior use in the literature, but due to the presence of closely related words in the scale, seemed to be genuine omissions from DSS:

1. *Yourselves* should be given the same score as all other reflexive pronouns. Lee omitted this particular pronoun while still scoring the singular form, *yourself*.
2. *Why* should be given the same score as other relative adverbs such as *where*, *when* and *how*. Again, this was left out of Lee's scoring rubric with no explanation, despite logically fitting into an existing developmental category. Even more oddly, *why* is used in an example of an incomplete relative clause by Lee, suggesting that a complete *why* clause should have been scored.

With these modifications to DSS decided upon, and with an output format chosen, it was now time to begin developing the application itself.

Chapter 3

Improving the Tagger

In order for a syntactic complexity analysis to be accurate, the underlying elements of syntax must also be accurately analyzed. Ideally, this requires accurately tagging the parts of speech in a sentence so that the syntactic structure is correctly interpreted.

Although some parsers are able to parse untagged texts, the ability to tag a text independently of parsing is still beneficial for a number of reasons. Using a pre-tagged text removes the need for a parser to have its own lexicon; as will be discussed in the following chapter, this also vastly simplifies the process of developing a new domain-specific parser. Furthermore, using machine learning to find part-of-speech tags for a lexicon is much more feasible than doing so to find grammatical relations; different treebanks can give entirely different parse trees for the same sentence, but tend to be relatively consistent in part-of-speech tagging.

The SYCORAX project inherited from CPIDR an unpublished tagger known as ODT, the Opportunistically Developed Tagger, developed by Michael Covington. Written in C# for the .NET 2.0 framework, ODT began with the most significant transformation rules obtained by Brill (1995) through a machine learning process on the Penn Treebank (Marcus et al., 1993, 1999). This rule set was further improved by Covington with the addition by hand of several new rules; many of these rules were not even attempted by Brill's algorithm due to their forms, which did not fit into Brill's limited set of templates. Yet ODT still had its share of inaccuracies, as I discovered while testing the tagger on a selection of sample sentences from Lee (1974); clearly, even more manual optimizations were necessary.

This ability to manually improve the tagger's rules is a significant advantage of Brill's tagger over the purely probabilistic taggers with which it was designed to compete. In a probabilistic tagger, the rule set consists of a large corpus of specific contexts, something which is not amenable to manual revision. In Brill's tagger, on the other hand, the result of the learning algorithm is a lexicon and a reasonably small set of transformations; it is quite easy to manually remove, modify, or add new transformations from this set as tagging errors are discovered.

3.1 The Testing Process

The usual measure of accuracy for part-of-speech tagging, as used by Brill among others, is the percentage of tokens from a tagged reference corpus which have been tagged correctly. The opposite of accuracy is *error rate*, the percentage of tokens which have been tagged incorrectly—i.e., 100% minus the accuracy. Changes in accuracy can be used to determine whether a new rule should be kept or discarded, both for manual optimization and automated learning: if it increases the accuracy, the rule is considered an improvement and kept, while if it decreases the accuracy, it is considered a regression and discarded.

As alluded to previously, the standard reference corpus for part-of-speech tagging is the Penn Treebank (Marcus et al., 1993, 1999), which was used in the training and testing of Brill's own tagger. The Treebank consists of four component corpora: the Brown corpus, comprising a variety of printed literature across various genres, both formal and informal; the Switchboard corpus, made up of transcripts of casual telephone conversations; the *Wall Street Journal* (WSJ) corpus, consisting of a selection of articles from its namesake newspaper; and a sample of the ATIS (Air Travel Information System) corpus, based on a prototype speech-recognition system for flight information. All texts within the Treebank have been manually tagged and parsed, thus providing an easy way to evaluate any tagger's accuracy.

To test the new manual optimizations to ODT, all of the Treebank corpora were used except for the ATIS sample; due to its small size, the artificiality of its language, and its domain-specificity, ATIS was negligible for the purpose of this project. The remaining three corpora provide a mix of formal written (Brown and WSJ), casual written (Brown), and spoken (Switchboard) forms of language, thus demonstrating the tagger's applicability to a wide variety of language use.

Although the percentage of correct tags over these three corpora was the primary measure used to determine the tagger's accuracy during this experiment, it was not the only measure. In several rare cases, described in further detail in the following sections, the accuracy in a specific context significantly improved with the addition of a rule, but the overall accuracy decreased at the same time; when this context was relevant to DSS, new rules were sometimes kept intact even despite the decrease in general accuracy.

During the testing process, several idiosyncrasies in the formatting of the Treebank data were discovered which were not defined in the accompanying documentation (Santorini, 1995). Specifically, it is possible for tokens to contain slashes and square brackets, both of which carry additional meaning in the Treebank format; the former are used to separate tokens from their tags, while the latter are used to delimit phrases. It was therefore necessary to ensure that the shell application for ODT only removed those slashes and brackets which were not part of a token. Additionally, as the documentation for ODT revealed, the lexicon was compiled with all slashes transformed to hyphens; thus, this same replacement also needed to be made for testing.

A number of typographical errors were present in the Treebank as well, and were corrected in the local copy used for testing. The Brown corpus contained eight stray curly braces which were not used as delimiters, as none of them had a matching opposite, but which also were not used as tokens; these were simply removed. The WSJ corpus contained two occurrences of the untagged token `Chiat\/NNP`, which should instead have been `Chiat\/Day/NNP` (where \/ is the standard representation for a token-internal slash).

Finally, one additional modification had to be made to the Switchboard corpus, due to a quirk of the tagging guidelines. This corpus was added in the third edition of the Treebank, while the written corpora dated back to the second edition; however, between the two editions, the tagging guidelines were changed so as to make distinctions which were originally left ambiguous. For just one example, the infinitive and prepositional uses of *to* received the same ambiguous tag in Treebank-2, but were tagged differently from each other in Treebank-3. The good news, however, is that converting texts from the newer tag set to the older one is trivial; Covington had already developed a tool (unpublished) to do exactly that, and it was this version of the Switchboard corpus on which tests were run. For reference, the original Treebank-2 tag set is reproduced in Appendix A.

3.2 Errors from Lee

The first set of optimizations to be proposed for ODT are summarized in Table 3.1; these consisted of nine new rules and four enhancements to existing rules. Those rules marked with an asterisk are especially significant, as these were already present in Brill's learned rule set (Brill, 1994); the remainder are logical extensions of existing rules based on errors found in Lee's examples.

Table 3.1 **Added and modified rules in ODT. Those marked with * were found in Brill's rule set.**

Added rules		
#	Transformation	Condition
1	*wan* → VBP	next token is *na*
2	*lem* → VB	next token is *me*
3	VBP → VB	one of the previous two tags begins with VB
4	VBD → VBN	one of the previous two tags is VBN
*5	POS → VBZ	next tag is DT
*6	POS → VBZ	previous tag is WP
*7	POS → VBZ	previous tag is DT
8	POS → PRP	previous token is *let*
*9	*so* → IN	next token is PRP or DT, or next token is *that*

Modified rules			
#	Transformation	Original	Modification
10	VBP → VB	previous tag is TO	one of the previous three tags is TO
11	VB → VBP	previous tag is PRP	previous tag is PRP, but not an object pronoun; moved above other VB/VBP rules
12	VBD → VBN	previous tag is PRP	same, but moved above other VBD/VBN rules
13	VBD → VBN	previous tag is VBD	one of the previous two tags is VBD

The modifications from Table 3.1 were initially added to the parser as a group, improving the overall accuracy by 0.0876%. Further improvements were introduced by selectively removing rules or parts of rules which were found to produce regressions. These iterations, and the resulting accuracies, are detailed in Table 3.2. With the most suitable set of modifications selected, it was possible to increase the accuracy of ODT by 0.0710% on the Brown corpus, 0.3967% on Switchboard, 0.0493% on WSJ, and 0.2166% overall.

Table 3.2 **Accuracy of modified versions of ODT. Rule numbers referenced are those from Table 3.1.**

#	Modification	% Accuracy				Revert To
		Brown	Swbd	WSJ	Overall	
1	Original ODT	94.9463	86.2832	95.4118	91.0411	
2	All modifications in Table 3.1	95.0100	86.4459	95.3965	91.1287	
3	Undo ordering from 12	95.0115	86.4462	95.3995	91.1301	
4	Undo rules 4 and 13	95.0153	86.4496	95.4476	91.1461	
5	Undo rule 3	95.0075	86.3882	95.4438	91.1145	4
6	Undo rule 9	94.9956	86.6036	95.4387	91.2104	
7	Restore *so that* from rule 9	94.9989	86.5945	95.4397	91.2073	6
8	Restore *so* `PRP/DT`	95.0121	86.4586	95.4466	91.1492	6
9	*so* `PRP` only	95.0085	86.4682	95.4438	91.1520	6
10	*so* `DT` only	94.9992	86.5941	95.4415	91.2076	6
11	Undo rule 11	94.9684	86.5715	95.4317	91.1865	6
12	Undo rule 10	94.9999	86.6129	95.4423	91.2168	
13	Modify rule 10 to look at only 2 prior tags	94.9994	86.6141	95.4421	91.2171	
14	Undo rule 5	94.9967	86.5526	95.4384	91.1868	13
15	Undo rule 6	94.9950	86.5848	95.4355	91.2005	13
16	Undo rule 7	94.9952	86.5067	95.4357	91.1643	13
17	Undo rule 8	94.9917	86.5776	95.4412	91.1980	13
18	Restore rule 13	94.9957	86.6094	95.4189	91.2075	13
19	Restore rule 4	95.0035	86.6148	95.4400	91.2180	
20	Modify rule 4 to look at only 1 prior tag	95.0053	86.6152	95.4437	91.2196	
21	Undo rule 11 ordering	95.0173	86.6799	95.4611	91.2577	
22	Undo rule 11 object pronoun	94.9871	86.6432	95.4536	91.2308	21
23	Modify rule 3 to look at only 1 prior tag	94.9930	86.5620	95.4362	91.1896	21
24	Add PDT as well as DT for rule 7	95.0173	86.6799	95.4611	91.2577	

One interesting pattern which deserves further notice, and which occurs again in further experiments, is that not all rules affect all corpora equally. The most notable example in this case is rule 9, which improves the tagger's accuracy on the two written corpora at the expense of both the Switchboard corpus and the Treebank as a whole. This discrepancy is largely due to the stylistic difference between the corpora: in colloquial speech, it is more likely that *so* is used as an adverb, which the addition of this rule limits. For accurate disambiguation of *so*, context beyond the set of immediately surrounding words is necessary, something that is not possible in Brill's tagger.

Although the overall increase in accuracy may seem minor, each of these improvements had a notable effect on DSS accuracy. *Wanna* and *lemme* were not identified as verbs whatsoever, thus causing any sentence with only these verbs to be ignored entirely; the same was also true regarding the contraction *'s*. The errors involving `VB` and `VBN` caused main verbs to be mistaken for secondary verbs and vice versa, sometimes with a drastic difference in point value. Finally, adverbial *so* is ignored entirely in DSS, while prepositional *so* is significant enough to earn five points.

3.3 The "Her" Ambiguity

Another error common to many Brill-style taggers, including ODT, was discovered by Deborah Keller-Cohen at the University of Michigan (personal communication, June 22, 2010). This error involves the word *her*: depending on surrounding context, it may be either a personal pronoun (*He likes **her***) or a possessive pronoun (*He likes **her** dog*). However, because no transformation rules exist to handle this ambiguity and because *her* occurred most often in the training corpus as a possessive pronoun, that tag is consistently given to the word by ODT even when it is not correct.

Although this distinction is mostly insignificant in DSS, where both forms of *her* are scored the same, it does still have its use in parsing; if an instance of *her* is unambiguously tagged as an object pronoun, it can safely be ignored in the rule attaching possessive pronouns as modifiers. In addition, as will be discussed later in this chapter, the methods used to improve this rule, and the lexical quirks discovered during its development, were also of use in developing rules which were more relevant to DSS.

In this case, the problem was that the obvious rule—transforming possessive pronouns (`PRP$`) into personal pronouns (`PRP`) except when preceding nouns or adjectives—produced a regression on 0.0268% of the Treebank as a whole. Further investigation of the resulting errors revealed the reason: if a noun was incorrectly tagged (e.g., identifying *walk* in *her walk* as a verb), the preceding pronoun would now be incorrectly tagged as well. Yet preventing this transformation when the following word *could* be a noun or adjective also led to an unexpected regression in the tagger's accuracy.

A closer look at the contexts of these new errors revealed the problem: some words in the lexicon have *extremely* rare probabilities of occurring as nouns or adjectives. In the Penn Treebank, for instance, the word *and* is tagged as a noun in the title *Jake and the Fatman*; similarly, *of* is tagged as a noun in *The University of Washington*. Even quotation marks can be tagged as nouns when used as an abbreviation for "inches." These tags are entirely correct in these specific contexts, of course, but the problem is that they are not applicable in a more general context.

This problem was exacerbated by the fact that ODT's lexicon, directly inspired by Brill's, contains no contextual information. A probabilistic tagger would be able to determine that *of* cannot be a noun except when surrounded by proper nouns, but as far as a Brill tagger is concerned, *of* is a perfectly plausible proper noun in any context. However, ODT's lexicon did include one significant improvement on Brill's, which turned out to be extremely useful for the purposes of this rule. While Brill's lexicon only identified which tags could potentially apply to a given token, ODT's also identified the number of occurrences of each tag for a given token. This property of the lexicon allowed for an additional restriction: a word could only be transformed to a noun or adjective if it occurred frequently enough as a noun or adjective. This would prevent words like *of* and *and* from being mistakenly identified as nouns, while allowing more common nouns to be properly transformed. An experiment was performed to find the threshold for each tag which produced the best overall accuracy; these results are summarized in Table 3.3.

After these thresholds had been incorporated into the rules, the accuracy levels reached 95.0679% for the Brown corpus, 86.6462% for Switchboard, 95.4552% for WSJ, and 91.2532%

Table 3.3 **Thresholds for part-of-speech tags following the pronoun in the `PRP$` → `PRP` rule, in the order tested in ODT's rule set.**

Tag	Threshold
`NN`	8[1]
`NNP`	15
`NNS`	1
`NNPS`	1
`JJ`	13
`JJR`	1
`JJS`	1

[1] Also explicitly ignoring quotation marks, which occurred 56 times tagged as a noun.

overall. This is still slightly lower accuracy than before the `PRP$` → `PRP` rule was introduced; however, the decrease was now a much smaller 0.0045% of the overall corpus, and the accuracy in fact improved on the Brown corpus.

Based on the results of this admittedly limited experiment, it seems as though this form of rule—based not only on context, but also on the probability of a word having a given tag—could be a useful addition to Brill-style tagging. Such rules extend Brill's tagger with some of the probabilistic enhancements of stochastic tagging, while still maintaining the predictability and the simpler lexicon of rule-based tagging. The threshold could be discovered using new rule templates; for instance, the class of rules learned in this experiment are all defined by the template "the current word is tagged z and the next word occurs at least x times in the lexicon as w," where z and w are part-of-speech tags and x is a bounded integer value. A transformation-based error-driven learning algorithm such as Brill's can then easily test various values to determine the combinations of tags and threshold which produce the best results.

3.4 Further Improvements

As testing of SYCORAX continued using a variety of sample texts, several additional classes of tagging errors were discovered, which are summarized below.

3.4.1 "Like" as a Verb

Another sentence included in Lee's DSS sample is *I like eating cookies.* This particular sentence, which was scored by SYCORAX as lacking a main verb, revealed yet another missing transformation in ODT. In the lexicon, *like* occurred most often as a preposition, yet only one applicable rule exists in Brill's trained tagger. This rule transforms `IN` to `VB` if the second word following the preposition is tagged `VB`; although this rule will correct sentences in which *like* is followed by an infinitive (e.g., *I like to parse*), it still fails to correct the very sentence that revealed this omission.

The development of a rule to correct the tagging of *like* is summarized in Table 3.4. In

Table 3.4 **Optimization of transformation rules for the word *like*. All rules change *like* to a `VBP` in the context provided, before any `VBP` → `VB` rules are processed.**

#	Context	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	No *like* transformation	95.0679	86.6462	95.4552	91.2532
1	No verb within the 3 prior words	95.0410	86.5882	95.4422	91.2157
2	No verb within the 2 prior words	95.0311	86.5642	95.4391	91.2011
3	No verb after prior punctuation mark	95.0499	86.5941	95.4465	91.2220
4	After *do* or `MD` + optional *not* or *n't*	95.0755	86.6785	95.4597	91.2714
5	4 + After `NNS/PRP` and no other verbs	95.0747	86.7056	95.4577	91.2833
6	Same as 5, but only using subject `PRP`	95.0748	86.7058	95.4577	91.2834
7	Same as 6, but for all verbs	95.0696	86.6921	95.4528	91.2743
8	Same as 6, but stopping `VB` scan at `IN/CC` as well	95.0719	86.6970	95.4551	91.2779
9	Same as 6, but scan whole sentence for `VB`	95.0761	86.6847	95.4594	91.2744
10	Same as 6, but include *does* and *did* too	95.0738	86.7091	95.4577	91.2847
11	Same as 10, but without `NNS/PRP` fix	95.0744	86.6817	95.4598	91.2727
12	Same as 10, but look for `VB` after as well	95.0736	86.6979	95.4583	91.2796
13	Same as 10, but allow 0 or more `RB` after `NNS/PRP`	95.0739	86.7153	95.4579	91.2877
14	Same as 13, but only allow one `RB`	95.0739	86.7152	95.4579	91.2876
15	Same as 13, but allow `RB` after *do*/`MD`	95.0738	86.7169	95.4582	91.2885

the end, the contexts which produced the best improvement were when *like* was preceded by a modal or a form of *do*, or when *like* was preceded by a plural noun or personal pronoun and had no preceding verb. In both cases, any number of adverbs could sit between *like* and the preceding word as long as the preceding context remained the same.

3.4.2 Adjectives and Adverbs

The next class of errors to be discovered involved adjectives and adverbs; the development process for this set of rules is shown in Table 3.5. With the exception of *very*, all of these modifications affected the Indefinite Pronouns score of DSS, which counts certain adjectives but not identically-spelled adverbs; the rule for *very*, on the other hand, was to prevent spurious attachment of *very* as a complement of a verb.

Table 3.5 **Optimization of adjective- and adverb-related rules.**

#	Modification	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	End of *like* iterations	95.07384	86.71688	95.45819	91.28848
1	Fix `JB` typo in `JJ` → `RB`	95.07230	86.74382	95.45812	91.30062
2	*very*/`JJ` → `RB` before `JJ`	95.07239	86.74410	95.45850	91.30088
3	Keep *very* change, undo `JB`	95.07393	86.71716	95.45858	91.28874
4	Restore `JB` fix; change *very*/`JJ` before `JJS`	95.07248	86.74420	95.45850	91.30095
5	Change *very*/`JJ` before `JJS` and `JJR`	95.07248	86.74420	95.45850	91.30095
6	Change *very* when tagged other than `JJ`	95.07248	86.74420	95.45850	91.30095
7	*most*/*least* → `RBS` before `JJ`x	95.07128	86.74438	95.45804	91.30060
8	*most*/*least* → `RBS` before only `JJ`	95.07137	86.74443	95.45804	91.30064
9	only adjust *most*/*least* after `DT`	95.07196	86.74420	95.45796	91.30066
10	only adjust *most*/*least* after `PRP$`	95.07222	86.74471	95.45913	91.30129
11	remove most/least fix; add `RB JJ` to `PRP` fix	95.07384	86.74504	95.45905	91.30184
12	add `RB RB JJ` to `PRP` fix	95.07376	86.74508	95.45913	91.30186
13	add `VBN` as well as `JJ` for `PRP$` rule	95.08153	86.74518	95.46440	91.30536
14	add `JJS RB` → `RBS RB` rule before `PRP$` rule	95.08008	86.74308	95.46254	91.30349
15	`JJS RB` → `RBS RB` only before `JJ`/`VBN`	95.08102	86.74518	95.46386	91.30507
16	`JJS VBN` → `RBS VBN`	95.08119	86.74532	95.46456	91.30538
17	same as 16, but add `JJR` too	95.08349	86.74564	95.46642	91.30664

The first of these errors was the result of a simple typo in ODT. As intended, one rule would have transformed any adjective (`JJ`) preceding an adverb (`RB`) into an adverb itself. As written, however, the rule transformed the nonexistent tag JB, which obviously had no effect. Correcting JB to JJ in this rule unexpectedly produced minor regressions on the Brown and WSJ corpora, due to constructions such as *answerable directly* and *unpopular domestically* which are uncommon in spoken language, but did produce the expected increase in accuracy on the Switchboard corpus and on the Treebank as a whole.

The next modification involved the word *very*. This word can act as either an adjective (e.g., *this* ***very*** *minute*) or an adverb (***very*** *happy*); however, ODT often failed to identify *very* as an adverb. In this case, the relevant contexts were as expected: *very* should be transformed to an adverb immediately before any adjective, either inflected or uninflected.

Another error concerned the words *most* and *least*. These were mis-tagged as superlative adjectives (JJS) in contexts where a superlative adverb (RBS) was correct (e.g., *not the* ***least*** *concerned*)—quite literally a seven-point distinction in DSS. This turned out to be the result of two separate omissions. The first was obvious: JJS needed to be transformed to RBS before an adjective or past participle. The second, however, was unexpected, and was a side effect of the pronoun transformation rule from Section 3.3. As discovered through further testing, this rule also needed to allow for one or two intervening adverbs between a pronoun and an adjective, as well as to allow for past participles in place of adjectives.

3.4.3 Modals and Novel Adverbs

Another set of errors involved the interaction of modals and verbs, along with one minor typo which had previously escaped notice. These are summarized in Table 3.6.

The error which prompted this set of modifications was that modals (MD) were sometimes mistakenly identified as uninflected verbs (VB) with the same spelling; for instance, *can* can also be a verb meaning “to put something into a tin.” This caused certain compound verbs to be incorrectly analyzed by the parser, which in turn caused the DSS analyzer to interpret them as strings of several main verbs. The rule which produced the greatest improvement

Table 3.6 **Optimization of modal- and verb-related rules.**

#	Modification	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	End of adjective/adverb iterations	95.08349	86.74564	95.46642	91.30664
1	Don't change `MD` if it precedes a `VB`	95.08495	86.74723	95.47037	91.30885
2	Change `MD`, but retag at end if precedes `VB`	95.10297	86.75175	95.48046	91.31836
3	Retag `MD` 1 or 2 places before `VB`	95.10715	86.75231	95.48092	91.31982
4	Fix typo in `MD`→`VB` rule; undo `MD` retag	95.07726	86.73324	95.45982	91.29743
5	Same as 4, but restore `MD` retag	95.10502	86.74830	95.47976	91.31708
6	Novel adverbs are `RB`, not `RR`	95.10502	86.74932	95.47976	91.31756

in the tagging of compound verbs, and thus in their parsing, transforms any suitable verb back into a modal if it occurs one or two words before an infinitive-form verb.

During the testing of this rule, two other typographical errors were also discovered in the tagger. An existing rule added by Covington to transform modals located after another verb to infinitive-form verbs (e.g., *He* ***did will*** *his computer to his heir*) failed to trigger when the prior verb was tagged `VBP`; this was clearly erroneous given the comments in the source code. Surprisingly, correcting this error as intended produced a minor regression, but as the transformation after `VBP` was clearly Covington's original intent, this correction was left intact. Another bug affected novel adverbs that did not occur in the Treebank, tagging any *–ly* word as the nonexistent tag `RR` instead of the correct `RB`; however, the improvement from this rule was negligible even on the Switchboard corpus, which was not used to train the lexicon.

3.5 Post-Processing

Unfortunately, this last set of modifications still did not solve all problems involving modals. In the sentence *Who will water the plants?*, for example, the tagger initially identified *water* as a noun; by the time *water* was finally transformed to a verb, it was already too late to

Table 3.7 **Addition of post-processing rules performed after the initial tagging pass was complete.**

#	Modification	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	Before post-processing	95.10502	86.74932	95.47976	91.31756
1	*that* → WDT	95.11305	86.75809	95.50264	91.33009
2	(PRP$) NN$x$ *n't* → (PRP) VBx *n't*	95.11305	86.75804	95.50264	91.33007
3	Same as 2, but with → MD first	95.11305	86.75804	95.50264	91.33007
4	Same as 3, but with VBP instead of VB	95.11305	86.75813	95.50264	91.33011
5	WRB POS → WRB VBZ	95.11399	86.75921	95.50272	91.33087
6	*n't* NNx → *n't* VBx	95.11697	86.78205	95.51311	91.34518
7	Retag best word as verb if none found	95.05608	83.83274	95.48627	89.94830
8	*gots* → VBZ	95.05608	83.83274	95.48627	89.94830
9	VB VBP → MD VB	95.05642	83.83348	95.48674	89.94886
10	All of the above fixes except 7	95.11732	86.78279	95.51358	91.34574

transform *will* to a modal. Again, this error caused an incorrect parse tree to be produced, which in turn caused the DSS analyzer to score the verb as two simple verbs rather than as a higher-scoring compound.

However, another option had presented itself in how SYCORAX was designed. The program contained an additional loop which performed a number of additional non-tag-related transformations on the tagged text after the initial tagger run but before parsing began, for the convenience of the parser. This stage can be seen as either post-processing or pre-processing, depending on whether it is considered relative to the tagger or parser; as this chapter is about the tagger, it will be referred to as post-processing here.

Nine new rules were proposed during the development of this post-processing stage, summarized along with their resulting accuracies in Table 3.7. The first of these rules retagged the word *that* as WDT—i.e., an interrogative determiner like *who* or *what*—when preceding a past participle (VBN) or a modal (MD), both of which were often transformed to their correct tags only after the tagger had left *that*. This distinction is important within DSS; the word *that* may be given three different scores depending on its function. Two other rules took advantage of properties of the contraction *n't* to identify verbs that were originally missed in the DSS analysis: the word which precedes must be a verb rather than a noun, and the word which follows must also be a verb unless the preceding verb is *be* or *have*. Yet another rule was designed to correct the tagging of the class of sentences described at the beginning of this section—e.g., *Who will water the plants?*—by transforming consecutive `VB VBP` to `MD VB`.

In addition to the rules described above, two further rules were discovered during the development of the post-processing stage. The first of these rules, created in response to SYCORAX not identifying the verb in a sentence with the contraction *where's*, transforms *'s* to a verb after an interrogative adverb; the other, in response to a drastically incorrect parse of a sentence containing the nonstandard verb form *gots*, transforms that word into a third-person singular verb. Unlike the remainder of the post-processing rules, these two rules were extremely simple, with no potential interference from other incorrect tags at all; this naturally prompted the question of whether any of the other rules could also work just as well in the main loop.

The obvious way to test this theory was to move each rule from post-processing into the main loop one at a time; the results of this experiment are shown in Table 3.8. Indeed, all but three of the rules produced the same result on the Treebank whether in post-processing or in the main tagger loop. These three were rule 9, the *Who will water the plants?* rule; rule 1, the rule to tag *that* as a `WDT`; and rule 6, the rule to transform nouns to verbs after *n't*. As expected, for reasons described earlier, leaving rule 9 in the main loop produced the same result as omitting it entirely; that left rules 1 and 6.

The regression which resulted from the movement of rule 6 to the main loop was actually due to a bug in an earlier rule described in Section 3.4.3. The retagging of modals located two words before another verb caused *need* to be tagged incorrectly in many cases, such as the coordination *need or want*. Adding an exception for *need* not only improved the tagger's

Table 3.8 **Movement of tagger rules from post-processing to the main tagger loop. Rule numbers referenced are those from Table 3.7.**

Rules Moved	% Accuracy			
	Brown	Swbd	WSJ	Overall
All but 7 and 9 in post	95.11697	86.78205	95.51311	91.34518
Moved to main loop:				
1 only	95.11621	86.78163	95.51218	91.34453
4 only	95.11697	86.78205	95.51311	91.34518
4 and 5	95.11697	86.78205	95.51311	91.34518
4, 5, 6	95.11706	86.78195	95.51311	91.34516
4, 5, 8	95.11697	86.78205	95.51311	91.34518
4, 5, 8; 6 modified to alter any possible verb	95.11552	86.77095	95.51366	91.33979
4, 5, 8 + ignore *need* _ `VB`	95.11749	86.78386	95.51552	91.34683
4, 5, 6, 8 + ignore *need* _ `VB`	95.11757	86.78386	95.51552	91.34685
4, 5, 6, 8, 1 in loop	95.11680	86.78345	95.51459	91.34620
4, 5, 6, 8, `VBN` 1 in loop; `MD` 1 in post	95.11757	86.78386	95.51552	91.34685

accuracy with rule 6 in post-processing, but caused the accuracy to *further* increase when that rule was moved to the main loop.

As for rule 1, the rule to transform *that* into a `WDT`, its problem was revealed when it was separated into two sub-rules. In this case, the problem was closely related to that for rule 9: some modals were not identified as such until later in the sentence, and so the transformation did not occur before modals. This was solved by leaving the transformation of *that* before modals in post-processing, but moving the transformation before past participles to the main loop.

3.5.1 The Verb Rule

One last rule shown in Table 3.7 was not previously discussed, but deserved further investigation. This was a special case to handle sentences where the tagger failed to identify a main verb, which in turn caused the sentence to be ignored in the DSS score. If no verb was found in the sentence, this new rule would find the best choice of word to transform to a main verb (`VBP`, `VBZ`, or `VBD`) and perform the transformation as appropriate.

The initial implementation of this rule produced a significant decrease in accuracy, the reason for which was a rather unfortunate oversight: the tagger would perform a transformation even if *none* of the words in the sentence could plausibly be a verb according to the lexicon. Even limiting the transformation only to those words which could be tagged as verbs still produced a regression, albeit not as large.

Much like the preposition rule described in Section 3.3, this rule was suited to an optimization problem based on the probability of a given word being a verb. The results of this experiment are summarized in Table 3.9, with the best overall accuracy found using a threshold of 33 occurrences as any form of main verb.

3.5.2 Further Optimizing Post-Processing Rules

The rule to transform *that* to a relative determiner before modals was still problematic. It seemed as if there should be a way to handle this transformation in the tagger itself rather than in post-processing; it followed that if modals were often incorrectly tagged as uninflected verbs, then the rule should also apply when *that* preceded an uninflected verb. As shown in Table 3.10, this change did in fact improve the tagger's overall accuracy, even when the rule was moved to the main loop. It also followed that *that* should be transformed before inflected verbs as well (e.g., *the mouse* ***that*** *roared*; *the dog* ***that*** *barked*), and indeed, this produced an even further improvement in the correct identification of relative *that*; still more of a boost in accuracy resulted from including this rule in post-processing as well as in the tagger itself.

Another concern involved the rule for identifying incorrectly tagged modals, in which, as mentioned previously, *need* was to be ignored when it occurred two words before an uninflected verb. It seemed likely that there were other modals spelled the same as infinitive verbs which would also be affected by this same rule. However, a test using all known modals

Table 3.9 **Optimization of the rule to retag the best word as a verb, using various thresholds for occurrences in the corpus as `VBP/VBZ/VBD`.**

Threshold	% Accuracy Brown	Swbd	WSJ	Overall
No retagging	95.11757	86.78386	95.51552	91.34685
5	95.11322	86.80922	95.50078	91.35343
10	95.11441	86.81076	95.50202	91.35480
15	95.11501	86.81109	95.50823	91.35684
20	95.11586	86.81244	95.50892	91.35788
21	95.11586	86.81249	95.50892	91.35790
22	95.11612	86.81267	95.51117	91.35868
23	95.11646	86.81267	95.51110	91.35875
24	95.11655	86.81290	95.51210	91.35916
25	95.11663	86.81295	95.51210	91.35921
26	95.11629	86.81295	95.51218	91.35914
27	95.11621	86.81356	95.51218	91.35940
28	95.11621	86.81356	95.51218	91.35940
29	95.11621	86.81356	95.51218	91.35940
30	95.11621	86.81356	95.51218	91.35940
31	95.11595	86.81360	95.51218	91.35936
32	95.11595	86.81356	95.51218	91.35934
33	95.11595	86.81356	95.51327	91.35964
34	95.11586	86.81356	95.51327	91.35962
35	95.11586	86.81328	95.51319	91.35947
36	95.11595	86.81067	95.51319	91.35827
37	95.11595	86.81067	95.51319	91.35827
38	95.11595	86.81067	95.51319	91.35827
39	95.11561	86.81039	95.51319	91.35805
40	95.11561	86.81043	95.51327	91.35810

Table 3.10 **Optimization of the rule transforming *that* to a `WDT`.**

#	Modification	% Accuracy Brown	Swbd	WSJ	Overall
0	+`VBN` in loop; +`MD` in post	95.11595	86.81356	95.51327	91.35964
1	+`VBN`, +`VB`, +`MD` in loop only	95.11535	86.81491	95.51249	91.35990
2	Same as #1, but with +`MD` in post too	95.11612	86.81533	95.51342	91.36055
3	+`VB`x, +`MD` in loop only	95.12133	86.85728	95.52963	91.38596
4	Same as #3, but with +`MD` in post too	95.12210	86.85770	95.53057	91.38661
5	+`VB`x, +`MD` in both loop and post	95.12312	86.86059	95.53212	91.38865

from the lexicon revealed that only one additional word should be ignored: *dare*, which only increased the overall accuracy by an incredibly small 0.00002%.

Finally, additional testing revealed that the rule to add a verb when none was found, as discussed previously in Section 3.5.1, was still not entirely sufficient. Specifically, in the sentence **How you open?*, the word *open* was still identified as an adjective rather than as a main verb. The problem in this case was that certain words occurred frequently in the lexicon as infinitives (`VB`), but not as plural verbs (`VBP`), despite the two forms being spelled the same in all cases but *be*; for instance, *open* occurred 111 more times as a `VB`, but only 9 times as a `VBP`. A reasonable solution to this problem was to add `VB` to the list of tags to search; then, if the word which occurred most frequently as a verb was tagged `VB` and was not *be*, it would be retagged as `VBP`. This change, of course, also meant that a new threshold had to be found for the rule, and as shown in Table 3.11, the best result was now found using a threshold of at least 34 occurrences.

3.6 Final Modifications

Now that the post-processing stage had been sufficiently optimized, it was time to add a number of final transformations in the main loop for various structures that were overlooked.

One such omission involved the word *so*, the significance of which was discussed in Section 3.2; specifically, the tagger more often than not tagged *so* as an adverb, even when the word was used as a preposition (e.g., *I parse* ***so*** *you don't have to*). This in turn produced a chain reaction with another rule, which transformed nouns after adverbs to verbs—thus also transforming nouns after prepositional *so*. The seemingly obvious solution was to transform *so* before any noun into a preposition (`IN`); however, although this improved the accuracy on the written corpora, it caused a regression on the Switchboard corpus, largely due to the colloquial use of *so* as an intensifier. As shown in Table 3.12, however, leaving *so* as an adverb while adding an exception for *so* in the verb transformation rule did produce an increase in accuracy on all three corpora.

Table 3.11 **Optimization of the rule to retag the best word as a verb, using various thresholds for occurrences in the corpus as a main or infinitive-form verb.**

Threshold	% Accuracy			
	Brown	Swbd	WSJ	Overall
No verbs	95.12517	86.83164	95.53483	91.37645
All but `VB`	95.12321	86.86059	95.53212	91.38867
20	95.11732	86.85611	95.52475	91.38303
25	95.11783	86.85686	95.52475	91.38350
30	95.11808	86.85789	95.52607	91.38442
31	95.11817	86.85807	95.52707	91.38481
32	95.11817	86.85807	95.52925	91.38541
33	95.11860	86.85807	95.53033	91.38583
34	95.11868	86.85817	95.53033	91.38589
35	95.11868	86.85817	95.53025	91.38587
36	95.11877	86.85560	95.53025	91.38470
37	95.11868	86.85560	95.53025	91.38468
38	95.11885	86.85574	95.53025	91.38478
39	95.11885	86.85546	95.53025	91.38465
40	95.11885	86.85546	95.53025	91.38465
41	95.11894	86.85546	95.53025	91.38468
42	95.11945	86.85584	95.53041	91.38502
43	95.11954	86.85598	95.53088	91.38524
44	95.11971	86.85598	95.53088	91.38528
45	95.11988	86.85607	95.53088	91.38537
46	95.11945	86.85607	95.53088	91.38526
⋮	⋮	⋮	⋮	⋮
50	95.11962	86.85598	95.53088	91.38526

Table 3.12 **Various fixes for the transformation of nouns to verbs after *so*.**

#	Modification	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	No modification for *so*	95.11868	86.85817	95.53033	91.38589
1	*so* → `IN` before `NN`x	95.12107	86.85239	95.53088	91.38396
2	don't convert to `VBS` after *so*	95.11877	86.85831	95.53080	91.38611
3	don't convert after *so*, *that*	95.11877	86.85831	95.53080	91.38611
4	don't convert after *so*, *that*, *as*	95.11877	86.85831	95.53080	91.38611

Another overzealous transformation was that which transformed nouns to verbs when followed by pronouns. The problem in this case was that the transformation occurred before any pronoun, even if the pronoun was an object pronoun that could not reasonably be followed by a verb. In the phrase *the man she likes*, for instance, *man* would be transformed to a verb because it preceded a pronoun; in turn, this caused the resulting parse tree to become nonsensical. As expected, limiting this transformation to contexts including object pronouns (i.e., *her*, *him*, *it*, *me*, *them*, and *us*) increased the tagger's accuracy, with an improvement of 0.07417% on the Treebank as a whole.

Yet another tagging error involved verbs occurring after pronouns, as discovered when *bit* in *It bit you* was tagged as a noun and thus ignored by SYCORAX. There was already a rule in the tagger to transform a potential `VBZ` (third-person singular present tense verb) after a pronoun, but similar rules did not exist for past-tense verbs or other present-tense verbs. The process of optimizing this rule summarized in Table 3.13; the best accuracy was obtained by transforming to a `VBP` if the word existed in the lexicon with that tag, or a `VBD` if it could have that tag but not `VBP`.

Table 3.13 **Optimization of the rule to transform nouns to verbs after pronouns.**

#	Modification	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	No modification	95.21033	86.95167	95.55725	91.46028
1	`PRP NN` $\rightarrow$ `PRP VBD`	95.21212	86.94995	95.55802	91.46015
2	`PRP NN` $\rightarrow$ `PRP VBP/VBD` (`VBP` only after pronouns that agree)	95.22613	86.99973	95.56974	91.49018
3	`PRP NN` $\rightarrow$ `PRP VBP/VBD` (whichever triggers first)	95.22980	87.05776	95.57563	91.51980

Yet another problem was discovered through the sentence *Her baby bear ate it*, in which the tagger identified *bear* as a verb and the parser misidentified it as the main verb. The development of this rule is summarized in Table 3.14, with initial implementation in post-processing but with an eventual transfer to the tagger itself. As finally developed, the rule transforms any VB following a singular noun or any form of adjective and directly preceding a verb or modal into an NN, or if this is not possible, into an NNP.

Table 3.14 **Further experiments on transforming verbs directly before verbs to nouns.**

#	Modification	% Accuracy Brown	Swbd	WSJ	Overall
0	Before modifications	95.22980	87.05776	95.57563	91.51980
1	NN VB VBx/MD → NN NN VBx/MD	95.23125	87.05800	95.57866	91.52113
2	Same, but allow any NNx for prior word	95.23091	87.05795	95.57881	91.52106
3	Same, but allow either NN or JJx for prior word	95.23134	87.05800	95.57866	91.52115
4	Same, but moved into main tagger loop	95.23160	87.05800	95.57897	91.52130
5	Try transforming to NN, then NNP	95.23160	87.05804	95.57897	91.52132
6	Try transforming to NNP, then NN	95.23108	87.05804	95.57897	91.52119

Another error occurred in the sentence *She said, "Sit there!"*, in which ODT tagged *Sit* as a proper noun (NNP) due to its preference for case-sensitive comparisons, in turn causing SYCORAX not to score the verb. This was in fact supported by the lexicon; an article in the *Wall Street Journal* was about a Chinese immigrant with the surname Sit. An initial attempt at optimization, shown in Table 3.15, was based on the frequency of the word occurring as a proper noun; however, although this worked for *Sit*, it failed to account for even more common names such as *Mark* and *Cook* due to their frequency. Thus,the process was started once again, this time using rules based strictly on the word's context, which would apply to all names equally; this is detailed in Table 3.16.

A final set of optimizations all involved rules which transformed nouns to verbs, either requiring restrictions on existing rules or addition of new rules; these are all summarized in Table 3.17. The second of these in particular deserves special mention, as it reveals the wide-ranging effects that a small human error in the training corpus can have on an automated tagger. In a single sentence in the *Wall Street Journal* corpus, the word *mature* was mistakenly tagged as a proper noun (NNP) when actually used as an infinitive-form verb (VB); this, in turn, caused all occurrences of *mature* to be tagged as a proper noun when preceding any other proper noun, as in *the bills will mature December 21*, due to a rule for compound proper

Table 3.15 **Optimization of the rule for transforming proper nouns (NNP) to imperative verbs (VB).**

#	Iteration	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	Before modifications	95.23160	87.05804	95.57897	91.52132
At start of sentence and before RB, IN, EX					
1	< 12 occurrences as NNP	95.23134	87.05842	95.57881	91.52139
2	≥ 2 occurrences as VB	95.23202	87.05837	95.57912	91.52163
At start of sentence and before any word listed in the lexicon as RB					
3	≥ 2 occurrences as VB	95.23254	87.05832	95.57928	91.52178
4	< 23 occurrences as NNP	95.23262	87.05842	95.57904	91.52178

Table 3.16 **Further optimization of the rule to transform proper nouns (NNP) to imperative verbs (VB), based solely on context.**

#	Iteration	% Accuracy			
		Brown	Swbd	WSJ	Overall
At start of sentence and...					
1	Before could-be-RB.	95.23211	87.05832	95.57641	91.52087
2	Before could-be-RB, except *of*.	95.23236	87.05832	95.57773	91.52130
All of the below at start of sentence and excepting *of*:					
3	Before RB/IN/EX	95.23083	87.05828	95.57827	91.52104
4	Before RB/IN/EX/DT	95.23330	87.05832	95.57858	91.52178
5	Before RB/IN/EX/DT/JJx	95.23330	87.05837	95.57858	91.52180
6	Before RB/IN/EX/DT/JJx/NNS	95.23279	87.05828	95.57431	91.52043
7	Before RB/IN/EX/DT/JJx/NN	95.23365	87.05837	95.57788	91.52169
8	Before RBx/IN/EX/DT/JJx	95.23330	87.05837	95.57858	91.52180
9	Before RBx/IN/EX/DT/JJx/TO	95.23347	87.05842	95.57858	91.52187
10	Before RBx/IN/EX/DT/JJx/TO/PRP	95.23433	87.05846	95.57881	91.52217
11	Before RBx/IN/EX/DT/JJx/TO/PRP/PRP$	95.23459	87.05846	95.57897	91.52228
12	Same as 11, but including *of*	95.23416	87.05846	95.57765	91.52180

nouns. Again, each of these rules is significant for the purposes of DSS, as verbs contribute to the score while nouns do not.

Table 3.17 **The final set of optimizations to ODT.**

#	Modification	% Accuracy			
		Brown	Swbd	WSJ	Overall
0	Last good tagger iteration	95.23459	87.05846	95.57897	91.52228
Miscellaneous rules					
1	*that* . → DT .	95.24296	87.18674	95.58292	91.58527
2	JJ NNP → NNP NNP only when JJ is capitalized	95.24355	87.18688	95.58440	91.58590
Restrictions on NN*x* → VB*x*					
3	Ignore NN*x* DT → VB*x* DT immediately after a DT	95.27302	87.20217	95.60177	91.60538
4	Same as #3, but also after a PRP$	95.27806	87.20534	95.60309	91.60850
5	Same as #4, but also after a PRP	95.27806	87.20534	95.60309	91.60850
NN*x* → VB*x* after WP					
6	WP NNS → WP VBZ	95.28019	87.20515	95.60635	91.60987
7	#6 + WP NN → WP VBP	95.27883	87.19532	95.60837	91.60551
8	WP NN → WP VBP only	95.27669	87.19551	95.60511	91.60414
9	WP NNS → WP VBZ, except when WP is *what*	95.28096	87.20641	95.60658	91.61072
10	#9 + WP NN → WP VBP, except when WP is *what*	95.28438	87.20739	95.61015	91.61304
11	#10 + WP NN → WP VBD, except when WP is *what*	95.28447	87.20744	95.61015	91.61309
12	#11 + *what*/WP NN*x* → *what*/WDT NN*x*	95.27968	87.20940	95.60286	91.61074
Further restrictions on NN*x* → VB*x*					
13	Ignore NN*x* DT → VB*x* DT immediately after JJ*x*	95.30121	87.21499	95.62178	91.62412
14	Same as #13, but also after a CD	95.30351	87.21797	95.62799	91.62783

Chapter 4

JED: "Just Enough Dependency" Parsing

Developmental Sentence Scoring, like all measures of syntactic complexity, is inherently based on syntactic structure as a whole, not just the parts of speech of individual words. Thus, improving the accuracy of SYCORAX's part-of-speech tagger is not sufficient to improve its DSS accuracy; it is also necessary, at least to some extent, to analyze the larger structures that make up the sentence. Even local context may not be enough: consider, for instance, the difference between **Her crying in there*, which is ungrammatical, and *I saw her crying in there*, which contains the same constituent but which is grammatical.

As further evidence that tagging is insufficient for DSS, Lively (1984) gives a selection of sentences which are prone to human error when DSS is scored by hand. Many of these structures are incorrectly analyzed precisely because human raters fail to consider the structure of the sentence beyond immediate context. For just two examples, compound verbs in which an auxiliary is deleted (e.g., *They were eating chicken and **drinking** tea*) are only identifiable by analyzing a long-distance dependency, while the distinction between complementing and adjunct infinitives (e.g., *I want him **to go** home*; *I passed the store **to go** home*) requires identifying the class of verb on which the infinitive depends.

Yet the state-of-the-art parsers which were initially considered for use in SYCORAX proved to be far too inefficient with respect to memory or execution time—or, for that matter, with respect to both. Thus, for this project, a new parser was developed known as JED: **J**ust **E**nough **D**ependency. This parser uses a simplified dependency grammar and a modified parsing algorithm to generate dependency trees that include the distinctions necessary for DSS, without any need for backtracking or non-determinism.

4.1 The Problem with Existing Parsers

For the purposes of this project, a number of freely-available "off-the-shelf" solutions for parsing had originally been considered. However, a cursory test of the candidates revealed them all to be unsatisfactory for use in SYCORAX; indeed, the most up-to-date parsers also seemed to be the least satisfactory for the purposes of this project. This paradoxical situation is the result of a classic tradeoff in computer science: accuracy versus efficiency.

The current state of the art in automated natural language parsing primarily owes itself to the shared tasks presented by the annual Conference on Natural Language Learning (CoNLL). Since 1999, the conference has presented a task each year in which researchers evaluate machine-learning systems against one another on a common set of real-world linguistic data. The 2006 and 2007 CoNLL tasks (Buchholz and Marsi, 2006; Nivre et al., 2007a) were dedicated to dependency parsing in a variety of languages; the former evaluated the ability of machine-learning systems to generalize dependency grammars from the given treebanks, while the 2007 task also evaluated their ability to learn dependency relations beyond the domain of the training data.

The CoNLL task uses dependency parsing because there exists a standard and simple metric for the accuracy of dependency parsers, something that is not available for phrase-structure parsing. As the 2006 CoNLL documentation explains (Buchholz and Marsi, 2006), the accuracy of a dependency tree can be determined by finding the number of words whose head and dependency type match those in a gold standard dependency tree for the same sentence; this measure is referred to as the labeled attachment score.

The problem, however, is that the CoNLL task has strictly focused on accuracy, while ignoring efficiency-related factors such as speed and memory use. The reports on the 2006 and 2007 shared tasks compare the competing parsers exclusively in terms of their accuracy, with no mention, much less any discussion, of efficiency. For generalized parsing, accuracy alone is a useful metric; however, for the purposes of automated DSS analysis, efficiency is of greater importance. Only a subset of syntactic relationships are significant in DSS analysis, and the rest may be analyzed incorrectly or even ignored entirely with no ill effect; on the other hand, it is important that the time and memory taken by the parser be minimal, so as to make the automated DSS analysis more economical than a human rater and competitive with non-parsing-based DSS analyzers.

4.1.1 Comparison of Existing Parsers

The efficiency problem is not merely a theoretical concern, either, as can easily be demonstrated through tests of the two best-reviewed and most popular parsers of the 2006 CoNLL task: MSTParser (McDonald et al., 2006) and MaltParser (Nivre et al., 2007b). The results described below are not aberrations; Søgaard and Kuhn (2009), for just one example, have found similar results in their own tests.

With respect to memory consumption, pre-trained dependency models in themselves provide enough insight. The authors of MaltParser have provided two parser models learned from the Penn Treebank, one of which was trained using a linear machine-learning algorithm and the other of which used a polynomial algorithm. The latter, though slower, still takes up a full 200 megabytes when uncompressed into memory; the former is faster, but uses over 600 megabytes. MSTParser does not provide a pre-trained model, but training it on the included 200-sentence sample produces a model that is 8 megabytes in size—which hardly bodes well for a model trained on the hundreds of thousands of sentences in the Treebank.

The second matter concerns execution time: given an already learned model, parsing times are still unsuitably slow for applications such as automated DSS. On a MacBook with a 2.4 GHz Core 2 Duo processor, MaltParser's slower model took approximately 39 seconds, using almost 400 megabytes of RAM, to analyze a relatively short text of 96 sentences comprising 586 tokens. The faster model took thirteen seconds—still relatively long for the given text—and used over 850 megabytes of RAM in the process. As expected, the difference grows exponentially with larger texts: a 4,600-sentence corpus took 19 seconds to parse using the faster MaltParser model, and a whole *ten minutes* to parse using the slower model.

MSTParser could not be accurately compared for general use due to the lack of a pre-trained model, but using the above-mentioned 200-sentence model, the parser took approximately 18 seconds to parse a different 200-sentence sample.

Worse yet, an additional performance issue would arise if either of these parsers were to be incorporated into SYCORAX. Both MSTParser and MaltParser are written in Java, and thus would either need to run in an external Java runtime or to be doubly virtualized using IKVM. The former case would hinder SYCORAX's usability, would eliminate its self-contained nature, and would introduce potential compatibility problems; the latter case, on the other hand, would lead to an even greater performance bottleneck than running under a native just-in-time compiler for Java.

4.2 Foundations for a New Parser

In a similar vein to that discussed by Søgaard and Kuhn (2009), the obvious solution to these speed and memory concerns was to "reinvent the wheel" by creating a new parser. The focus of this parser was to be on efficiency rather than accuracy; although it is less accurate in general than the state of the art, the aim was to have equal or better accuracy on those structures that are of interest to SYCORAX, as measured via the accuracy of the resulting DSS scores.

4.2.1 Phrase-Structure versus Dependency

The first question regarding this new parser was whether it should be based on phrase-structure grammar or dependency grammar.

Abney (1991) describes the foundations of a simplified form of phrase-structure parser, the *chunk parser*. This type of parser breaks the parsing process down into two stages: a *chunker* which breaks the sentence down into phrases and analyzes the internal structure of these phrases, and an *attacher* which determines the connections between these chunks.

This initially looked like a suitably simple approach to take, but it quickly became apparent that it was not so simple as it looked. Although the set of rules given by Abney in his example grammar is relatively simple, the parsing algorithm Abney uses is a nondeterministic LR shift-reduce parser—not the simplest sort of parser to implement. Additionally, chunking—the easier part of the analysis—was not sufficient in itself for DSS scoring; it would be necessary to attach the chunks in order to determine certain necessary relationships such as subject and object. This requires significantly more lexical information, which, of course, is beyond the scope of Abney's paper.

Indeed, identifying subjects and objects was a problem with any phrase-structure parsing algorithm. For an example of why this is the case, consider the parse tree for *The dog chases the cat* generated by a phrase-structure grammar, as shown in Figure 4.1. Identifying the subject of *chases* requires finding the head descendant of the sibling of the parent of *chases*—or, in other words, the cousin of *chases* which is a noun. Determining this relationship becomes even more complicated when a prepositional phrase is also added to the subject, thus adding an additional level of depth to the tree as in Figure 4.2.

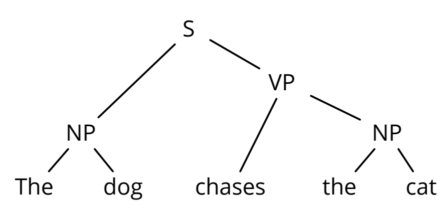


Figure 4.1 **A traditional parse tree for *The dog chases the cat.***

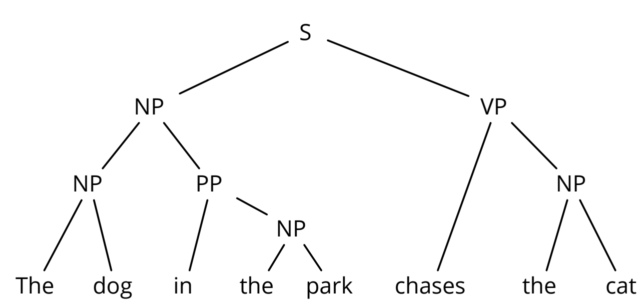


Figure 4.2 **A traditional parse tree for *The dog in the park chases the cat.***

Dependency parsing, in contrast, is well suited to finding the relationships necessary for accurate DSS analysis. Figures 4.3–4.4 show the dependency trees generated using one popular dependency grammar (Järvinen and Tapanainen, 1997) for the same two sentences shown in Figures 4.1–4.2. Note that in both of these sentences, finding the subject of *chases* is as simple as following a single branch of the tree, the only direct dependent of *chases* labeled `Subj`.

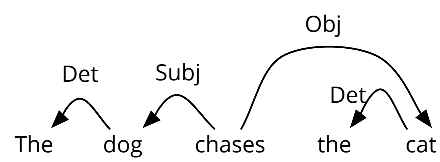


Figure 4.3 **A flattened dependency tree for *The dog chases the cat.***

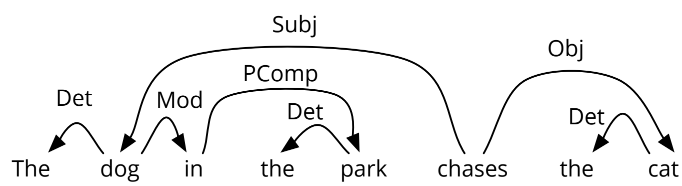


Figure 4.4 **A flattened dependency tree for *The dog in the park chases the cat.***

Thus, despite the fact that DSS was originally based on phrase-structure grammar, syntactic structures that are significant in DSS are ironically made much more apparent using dependency grammar.

4.2.2 The Parsing Algorithm

As discussed previously in this chapter, however, the state of the art in dependency parsers was far too complex in its implementation. What was needed for SYCORAX was something much simpler, going back to the basics of dependency parsing. Because the aim of this project was to create a dependency parser that focused on the relationships necessary for DSS, it was given the name JED: **J**ust **E**nough **D**ependency.

Covington (1990) provides an implementation of one reasonably simple algorithm, which has been further expanded in Covington (2001). The dependent-first variation of this algorithm is summarized in Algorithm 4.1 below.

Algorithm 4.1 A dependent-first algorithm for dependency parsing.

```
for each word W in sentence do
    for each head D preceding W, starting with the closest do
        if D can depend on W then
            Link D as dependent of W
        end if
    end for
    for each word H preceding W, starting with the closest do
        if W can depend on H then
            Link W as dependent of H;
            Break out of for loop
        end if
    end for
end for
```

In short, this algorithm iterates through each word of the sentence and, at each word, runs two loops on the set of preceding words. The first loop searches only those preceding words that have no head, attaching any of these words which could be a dependent of the current word as such. The second loop searches the full set of preceding words, marking the first suitable word, if any, as the head of the current word, and stopping there. A head-first version of the algorithm also exists, which simply swaps the order of these two inner loops.

For further efficiency, Covington (2001) describes an enhancement of the algorithm in which a list of the known heads and a list of all prior words are maintained in the course of the loop, and in which the inner **for** loops iterate through these two lists, respectively, so that no extra effort is wasted looking at prior words to determining whether they are heads. As will be discussed later, however, this alternative approach may not be necessary depending on the data structure used to represent the dependency tree.

Another variation on the dependency parsing algorithm discussed in Covington (2001) enforces projectivity, a property of dependency trees stating that branches of the tree cannot cross. This restriction is beneficial for languages such as English with a relatively fixed word order, as it acts as a further safeguard against incorrect long-distance dependencies and largely eliminates the need to write adjacency requirements into the grammar itself. There are several structures, such as fronted prepositional complements and standards of comparison, whose analysis may require overlapping dependencies in some grammars, as shown in Figure 4.5; however, other grammars are able to represent these same structures while maintaining projectivity, as shown in Figure 4.6. Hudson (1989) describes a simple grammar for English which fully enforces projectivity along with an algorithm to parse it, and Covington (1990) demonstrates that his own projectivity-enforcing algorithm is equivalent to Hudson's.

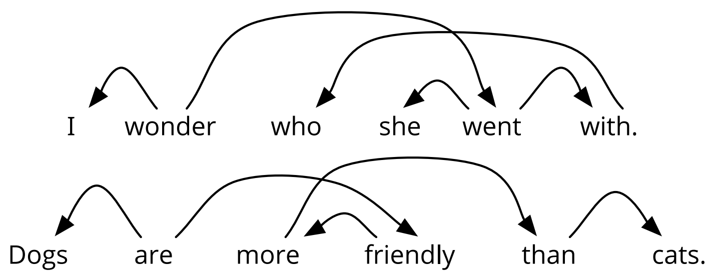


Figure 4.5 **Projectivity-violating dependency graphs for two valid sentences, using the grammar of Järvinen and Tapanainen (1997).**

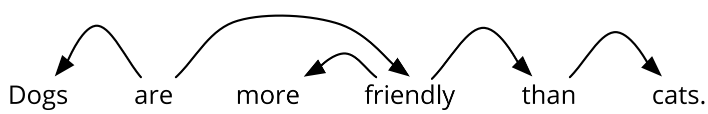


Figure 4.6 **An alternative, projective structure for *Dogs are more friendly than cats*, as implemented in JED's grammar.**

Covington's modified algorithm ensures projectivity by restricting the set of prior words which are searched in each loop so as to prevent overlap. In the loop to find dependents of the current word W, only the nearest *consecutive* members of the set of prior heads are tested; that is, if a word is a head but cannot be a dependent of W, the search stops there. In the loop to find a head for W, only the word preceding W and its ancestors are considered as possible heads.

This latter restriction, however, does not work as Covington intended, and in fact prevents some projective sentences from being analyzed. Consider the previously mentioned sentence *The dog chases the cat*, which, as seen in Figure 4.3, has no overlapping branches. Figure 4.7 shows the state of the dependency tree just before the parser has begun the "search for head" loop on *cat*. The word preceding *cat* is *the*, which cannot be a head of *cat*. The parser will then consider all ancestors of *the*—but the only ancestor is *cat*! Thus, using this version of the algorithm, it is impossible to attach *cat* as a dependent of *chases*.

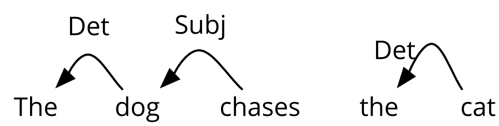


Figure 4.7 **The state of the parse tree for *The dog chases the cat*, before attaching *cat* as a dependent of *chases*.**

A solution to this problem is to set the initial value of H to the first word preceding W which does not directly or indirectly depend on W. Since all words between H and W are rooted at W, no overlap will occur; the path from H to W can be drawn above the entire subtree rooted at W. Thus, in the case of *The dog chases the cat,* H would initially be *chases* using this modified algorithm, and the object would be properly attached to the verb. A pseudocode version of this algorithm, as implemented in JED, can be seen in Algorithm 4.2.

4.2.3 Creating a Data Structure

With an algorithm finally chosen, it was now necessary to design a suitable data structure for the dependency tree. What made this task easier was the uniqueness constraint described in Covington (2001): each word can only have one head.

To represent an individual relationship within the dependency graph, a class known as `Dependency` was implemented. This class includes two members: `Type`, an instance of the `DependencyType` enumeration identifying the dependency label, and `HeadIndex`, an integer representing the index of the head word in the sentence.

Algorithm 4.2 The projectivity-preserving parsing algorithm used for JED.

for each word W in sentence **do**
  $D \leftarrow W - 1$
  ▷ Search last contiguous set of heads for dependents of W
  while $D \geq$ first word of sentence **do**
    if D has a head **then**
      $D \leftarrow D - 1$
      Go to start of **while** loop
    end if
    Test for dependency "$W \rightarrow D$"
    if dependency does not apply **then** ▷ It is a head, but not attachable
      Break out of **while** loop
    else
      Link D as a dependent of W
    end if
    $D \leftarrow D - 1$
  end while
  $H \leftarrow W - 1$
  ▷ Find nearest word that is not a dependent of W
  while $H \geq$ first word of sentence **and** W is an ancestor of H **do**
    $H \leftarrow H - 1$
  end while
  while $H \geq$ first word of sentence **do** ▷ Then check it, and all ancestors.
    Test for dependency "$H \rightarrow W$"
    if dependency does not apply **then**
      if H has no head **then**
        Break out of **while** loop
      else
        $H \leftarrow$ head of H
        Go to start of **while** loop
      end if
    else
      Link W as a dependent of H
      Break out of **while** loop
    end if
  end while
end for

That leaves identifying the dependent side of the relationship. Because each word can only have one head, the obvious choice is to represent the tree as an array of `Dependency` objects, with each element of the array representing the dependency with the word of the corresponding index as dependent. Using this representation, movement toward the head of the dependency tree can be performed in constant time for each step. Checking the type of dependency relationship between two words can also be done in constant time, as can determining whether or not a given word is a head.

The major performance barrier in this representation results from moving downward in the tree (i.e., from heads to their dependents); however, even this can be done in $O(n)$ time on the length of the sentence for each step, as it is simply a matter of scanning through the array and finding which dependents have the specified head. Performance could further be improved by adding a second array, with each element being a list of dependents, so that the set of dependents of a given word can be determined in constant time; however, given the length of most sentences, $O(n)$ is not a significant barrier.

4.3 Choosing a Grammar

The algorithm, of course, is the easier part of a parser to develop; the greater challenge lies in developing a grammar to identify the possible relationships among words.

Many modern parsers, such as the aforementioned MaltParser and MSTParser, have used machine-learning algorithms trained on manually-parsed corpora to generate their grammars. MaltParser, for instance, uses support vector machines, while MSTParser uses an online learning algorithm. While the use of machine-learned grammars does lead to improved accuracy, it also has a number of downsides: the results are unpredictable in comparison to an explicitly rule-based approach, and the learned model can quickly become large and time-consuming to an extent that even the most complicated sets of explicit rules do not.

This is largely aided by the purpose of this particular parser. As discussed previously, the intent of JED is specifically to identify those relationships that are significant to DSS;

errors and omissions elsewhere are negligible, as long as they do not affect the parsing of the relevant dependencies. What is important, in other words, is not the accuracy of the parse tree as a whole, but rather the accuracy of the resulting DSS score. With this restriction in mind, the problem of developing a rule set becomes much more tractable. Indeed, as discussed below, it was possible to manually develop a set of rules suitable to this task based on a relatively small corpus of sample sentences, with a few minor yet novel additions to the parsing algorithm.

The basis for this grammar was that of Järvinen and Tapanainen (1997). Although the authors did not document the full set of rules which comprised their implementation, their documentation does include sufficient detail to reconstruct a simpler version of that grammar. All types of dependencies are not only clearly defined, but also shown in context within dependency trees for actual sentences. For a few rules, it was necessary to slightly modify the rules as documented, for reasons such as the above-mentioned non-projectivity; these exceptions will be described in further detail later in this chapter. In addition, a number of dependency types were combined in the case of distinctions that are unnecessary for DSS; these will also be described in further detail below.

4.4 Handling Ambiguity

Development of the rule set began surprisingly well, but ran into a roadblock early in development due to several ambiguities that could not be handled by Covington's algorithm alone.

Consider the sentences *I want this* and *I want this dog*. In both of these sentences, *this* is tagged as a determiner; however, as shown in Figure 4.8, *this* is itself an object of the verb in the former sentence, while in the latter, *dog* is the object and *this* is a dependent of *dog*. Using Covington's algorithm verbatim, it is impossible to attach *this* to *want* at some unambiguous later point in the sentence; however, if *this* is attached to *want*, it will be impossible to later attach it to *dog* because of the uniqueness constraint. An almost identical ambiguity applies to the head-first prepositional dependency and tail-first infinitive dependency of the word *to*, as shown in Figure 4.9.

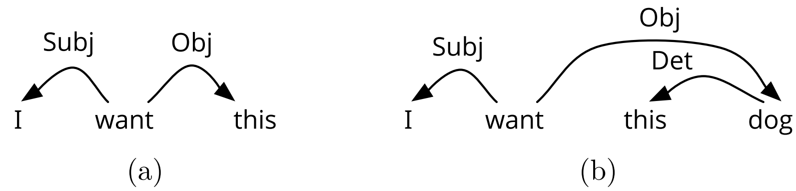


Figure 4.8 **Ambiguity of rules for determiners that can also serve as objects. In (a), *this* is an object of *want*, while in (b), *this* modifies the following word *dog*.**

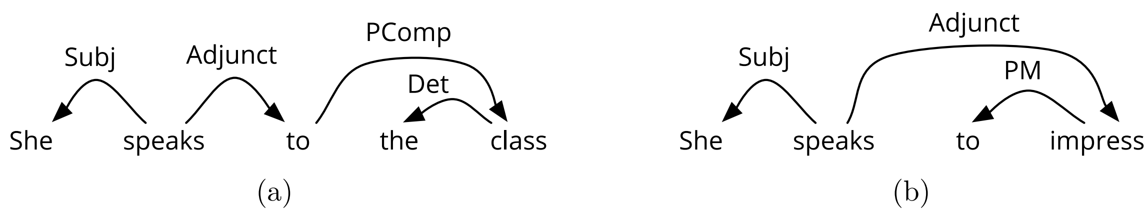


Figure 4.9 **Ambiguity of rules involving the word *to*. In (a), *to* is a preposition on which the following noun depends, while in (b), *to* is an infinitive marker that depends on the following verb.**

It would, of course, be possible to add backtracking into the parser, as Covington (2001) suggests for such ambiguities; although it is not automatically provided in C# as it is by Prolog, backtracking could still be implemented in the former by means of recursion. Another alternative was to use a non-deterministic approach, in which the parser held several alternative parses in memory at once. However, each of these had a disadvantage; along with the details of implementation, both of these alternatives would multiply the amount of time and memory taken during a parse by an additional $O(n)$. I was curious as to whether there were any potential ways of handling such ambiguities which were less time- and resource-intensive.

4.4.1 An Initial Attempt: Multi-Pass Parsing

The first approach to be tested involved the use of multiple passes of the parser, each of which encapsulated a subset of the full grammar of dependency rules. This way, it would be possible to postpone the analysis of one dependency until another had been fully analyzed; for instance, the tail-first dependency for determiner dependents could be analyzed in the first pass, while the head-first dependency could be analyzed in a second pass. The worst-case scenario for backtracking, as mentioned in Covington (2001), is $O(n^3)$; the worst case for a multi-pass parse, on the other hand, would be $p \cdot O(n^2)$, where p is the number of passes.

This multi-pass approach was easily implemented through a feature of C# known as delegates, a class of object which essentially acts as a pointer to a function. Using this construction, each subset of rules could be implemented as a separate function on the head and dependent; then, Covington's algorithm could run each pass with a different rule set as its parameter, requiring no unnecessary duplication of code. The advantage of this approach was that it was easily expandable to any number of passes.

The addition of a second pass proved to be useful for handling rule conflicts involving determiners, *to*, and even object and subject pronouns. A third pass was necessary to correct the precedence of verb chains over infinitives (e.g., *to have been going*) and compound nouns over object attachment (e.g., *of the English language*). A fourth pass was eventually added to solve a problem of ambiguity between interrogative subjects (*Who is he?*), fronted interrogative objects (*What did he do?*), and relative subjects of subordinate clauses (*I wonder what he did*).

Yet even this multi-pass approach was not enough to disambiguate certain constructions. The first ambiguity which made this limitation apparent involved prepositions which can also act as subordinating conjunctions. Consider the word *after*: as shown in Figure 4.10, it can introduce a prepositional phrase, as in *He left after the dog*, or a subordinate clause, as in *He left after the dog barked.* Here, *dog* cannot be attached as a complement of *after* until the parser has determined that *after* is not a subordinate conjunction of an even later verb. For this to work, however, attachment as a preposition must occur after the attachment of subjects due to the projectivity restriction; this in turn prevents *The scene after the* *intermission was the best*, as shown in Figure 4.10(c), from being parsed correctly, as the preposition attachment must occur before the subject attachment.

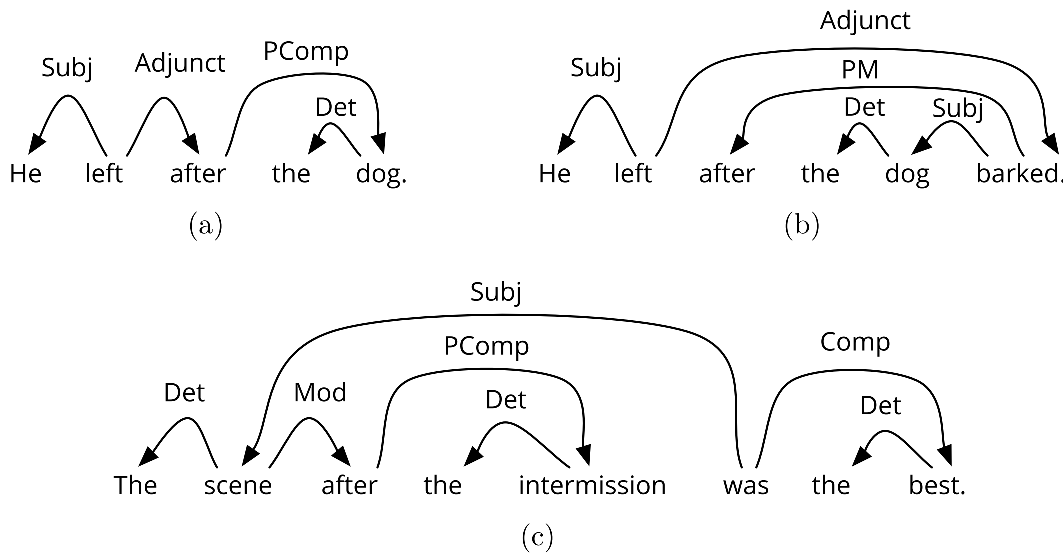
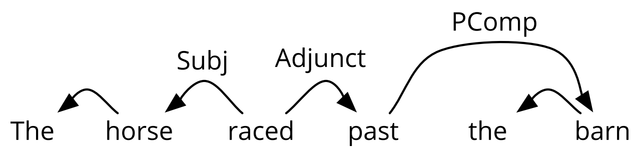


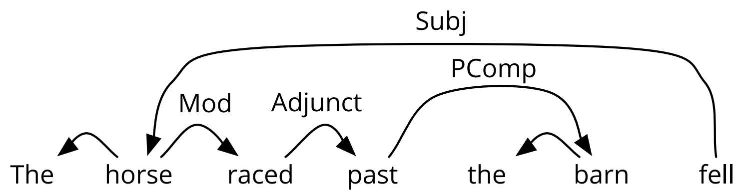
Figure 4.10 **Ambiguity of rules involving prepositions and subordinating conjunctions. In a multi-pass parse, if (b) can override (a), (c) will be impossible to parse due to the projectivity restriction.**

Another ambiguity which could not be handled by the multi-pass parser was that between past-tense verbs and past participles. For regular verbs, these two verb forms are spelled identically, and are thus often indistinguishable to a tagger; any regular past-tense verb (`VBD`) may actually be a past participle (`VBN`), and vice versa. This ambiguity is the reason that *The horse raced past the barn fell* is such an infamous "garden-path sentence" in English: readers initially interpret *raced* as the main verb of the sentence, realizing only after reaching *fell* that *raced* is actually a past participle modifying *horse*. The parse trees for this ambiguity, both before and after the attachment of *fell*, are depicted in Figure 4.11.

In this case, the problem is that the same pair of words (*horse*/*raced*) may be involved in either a head-first or a tail-first dependency with one another, and that the latter dependency follows the same rule as that of *horse* on *fell*. There is thus no suitable ordering of rules which can parse both of these sentences without backtracking: *horse* can only be attached to *fell* if *raced* has already been attached to *horse*, but *raced* cannot be prematurely attached as a dependent of *horse* because it may be the main verb instead.



(a)



(b)

Figure 4.11 **The expected dependency trees for *The horse raced past the barn fell.* (a) shows the state before, and (b) after, *fell* has been analyzed.**

For a DSS analysis to be accurate, it is necessary to analyze both of these structures correctly. If *horse* is not attached as a subject of *raced* in *The horse raced past the barn*, the DSS analyzer will identify the sentence as ungrammatical due to lack of subject-verb agreement. On the other hand, if *horse* is wrongly attached as a subject of *raced* rather than *fall* in **The horse raced past the barn fall*, the sentence will mistakenly be identified as grammatical. Clearly, a different approach was necessary.

4.4.2 Pseudo-Heads and Pseudo-Backtracking

The solution which was finally implemented was based on the observation that some dependencies are clearly unambiguous, while others remain ambiguous. For instance, the attachment of a subject to a verb is typically unambiguous, with the aforementioned exception of regular past-tense verbs. The attachment of an object to a verb, on the other hand, may be ambiguous: for instance, consider the sentence *I saw the green house paint*, where each word of *green house paint* may be an object of *saw* with *the* as its dependent.

For unambiguous dependencies, Covington's algorithm would run as usual. For ambiguous dependencies, on the other hand, the uniqueness constraint could be relaxed, allowing a dependent to be reanalyzed and moved to a later head during the "find dependents" loop. At the same time, the adjacency restriction would still apply to ambiguous dependents during the "find heads" loop. This essentially turns the dependents of ambiguous dependencies into *pseudo-heads*, invisible in the "find dependents" loop but still visible in the "find heads" loop. This behavior essentially allows for reanalysis in a manner similar to backtracking or non-determinism, but with only a single parse tree in memory at any given time.

Modifying Covington's algorithm, as implemented in JED, to handle ambiguous dependencies was relatively painless:

- An additional attribute named `Final` was added to the `Dependency` class; this would be set to true within a rule if a dependency is unambiguous, or false if it is in any way ambiguous.

- In the first **while** loop of Algorithm 4.2, "**if** D has a head" was changed to "**if** D has a head **and** D is a final dependent," allowing the dependents of ambiguous dependencies to be re-analyzed.

This modification, together with a suitably modified set of rules, allowed the ambiguous preposition *after*, as shown in Figure 4.10, to be successfully disambiguated.

Yet this was still not enough to handle *The horse raced past the barn fell*. Figure 4.12(a) shows how the sentence is parsed up to *fell* in a parser that allows for ambiguous dependencies. Using the modified algorithm, it is possible to detach *horse* from *raced* and reattach it to *fell*—but this, in turn, leaves *raced* stranded, as shown in Figure 4.12(b). To handle the reattachment of *raced*, it is also necessary to add the potential for side effects to parser rules; for instance, the rule attaching a noun to a verb could have a side effect that reversed any existing dependency with that noun as its dependent, thus producing the parse tree from Figure 4.11(b) as a result instead.

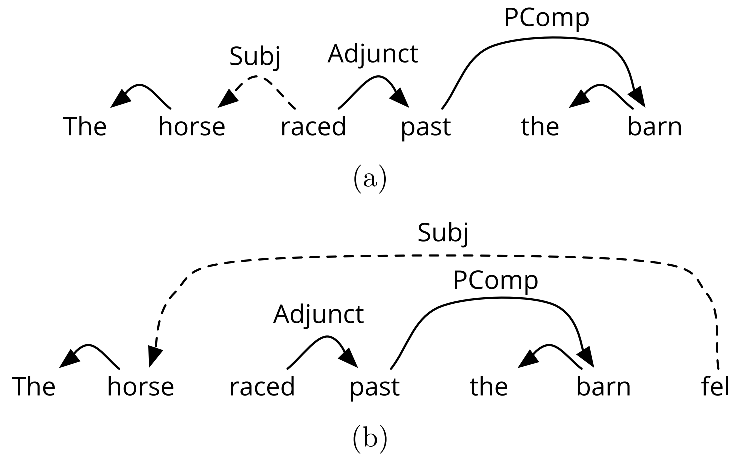
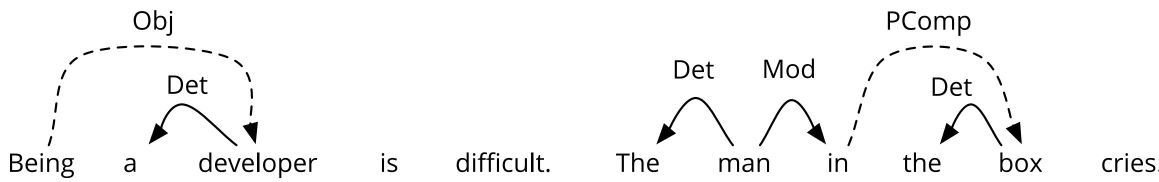


Figure 4.12 **The dependency trees for *The horse raced past the barn fell*, as generated using ambiguous dependencies without side effects. Dotted lines represent ambiguous dependencies, while solid lines represent final dependencies. (a) shows the state before, and (b) after, *fell* has been analyzed.**

This addition of side effects essentially behaves as *pseudo-backtracking*. Like backtracking, it allows earlier dependencies in the parse tree to be revised when later dependencies show the original analysis to be unworkable. Unlike backtracking, however, it is not necessary to entirely revert the parse to an earlier point; instead, only the relevant dependencies are modified, while others are left alone.

As new rules continued to be added, it became apparent that the parsing algorithm was still incomplete. Sentences such as *Being a developer is difficult* and *The man in the box cries* were not fully parsed; as shown in Figure 4.13, the parsing algorithm would stop looking for dependents of *is* and *cries*, respectively, after the attachment of the closest pseudo-head failed. In short, the search for dependents, instead of breaking out of the loop upon failed attachment of a pseudo-head, needed to restart the loop to search only for real heads. As shown in Algorithm 4.3, this was implemented through the use of an additional *if* clause which caused the loop to restart only for real heads if no pseudo-head matched; this allowed the two sentences from Figure 4.13 to be parsed correctly.



(a) (b)

Figure 4.13 **Two sentences which the original ambiguity-accommodating parsing algorithm failed to parse. In both cases, the real head which precedes the nearest pseudo-head is the actual subject.**

Algorithm 4.3 The "find dependency" loop, as modified to deal with ambiguous dependencies.

```
A ← true                                  ▷ Identifies whether to consider ambiguous dependents
while D ≥ first word of sentence do
    if D has a head and (D is a final dependent or A = false) then
        D ← D − 1
        Go to start of while loop
    end if
    Test for dependency "W → D"
    if dependency does not apply then           ▷ It is a head, but not attachable
        if A = true then
            A ← false
            D ← W − 1
            Go to start of while loop
        else
            Break out of while loop
        end if
    else
        Link D as a dependent of W
    end if
    D ← D − 1
end while
```

4.4.3 Bugs Unique to Pseudo-Backtracking

For the most part, this algorithm worked as intended with no problems, even after the addition of pseudo-backtracking. However, three significant problems were discovered which required slight modifications to the algorithm.

Cycles

The first of these problems was that, although the algorithm in itself could not produce a cycle, it was possible to inadvertently generate a cycle through the use of side effects—thus introducing the potential for the parser to become trapped in an infinite loop while traversing the parse tree.

Consider, for instance, the following toy grammar:

- `VB`x $\rightarrow$ `PRP` (`Subj`), if the head has no `VCh` dependents
- `VBG` $\rightarrow$ `VBZ` (`VCh`), with the side effect of making the new head a dependent of the tail's original head
- `VBP` $\rightarrow$ `VBZ` (`Adjunct`)
- `VBP` $\rightarrow$ `VBG` (`Adjunct`)
- `VBG` $\rightarrow$ `VBP` (`Adjunct`)

Now consider the sentence *I know she is trying*, tagged as *I*/`PRP` *know*/`VBP` *she*/`PRP` *is*/`VBZ` *trying*/`VBG`. Figure 4.14 shows the state of the parse tree at three different points during the parse. Figure 4.14(a) shows the parse tree immediately after running both loops on the word *is*; here, *know* has just taken *is* as a dependent.

After finishing with *is*, the parser then looks at possible dependency-first attachments headed by *trying*. *Trying* can have *is* as a dependent, as part of a verb chain, and *is* is currently a pseudo-head; attaching *is* to *trying* triggers a side effect making *trying* the dependent of *know*, as shown in Figure 4.14(b). The parser then looks at other possible dependencies headed by *trying*, and attaches *know* as a dependent of *trying*, thus creating a cycle, as shown in Figure 4.14(c). Without the addition of the side effect, this cycle would have been impossible: the parser would only search for possible heads of *trying* after *know* had already been made a dependent of *trying*.

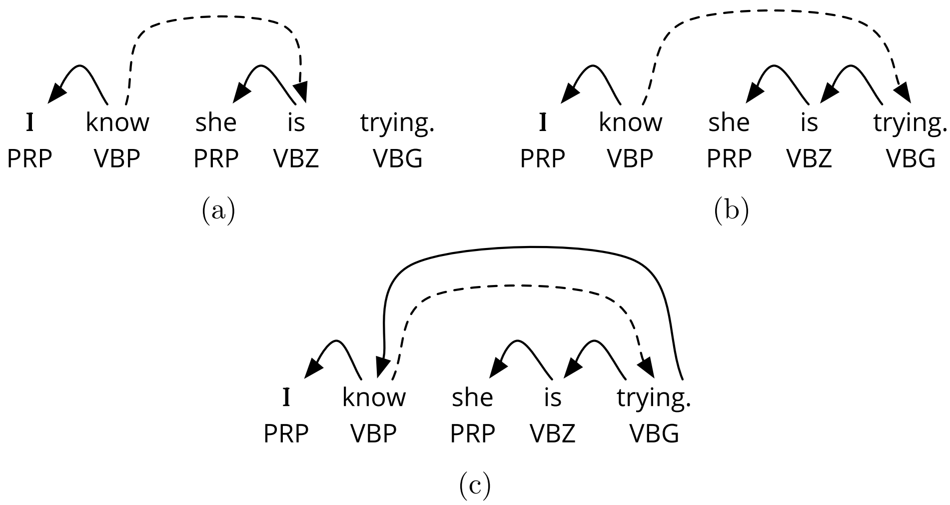
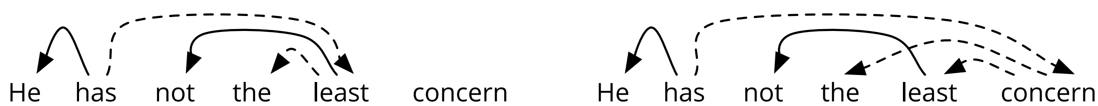


Figure 4.14 **The process by which a cycle is created in the sentence *I know she is trying*, using the toy grammar on page 66.**

To deal with this potential for cycles without any additional restrictions on the grammar, a method named `FixCycle` was added, which runs after the addition of each new dependency. This method determines whether the newly added dependency has produced a cycle, and if so, removes the existing dependency that has the new dependent as its head. In Figure 4.14(c), for instance, *know* $\rightarrow$ *trying* would be removed after *trying* $\rightarrow$ *know* was added. Although this method technically runs in $O(n)$ time, the average run time tends to be significantly less; the method only runs at all if the head of the new dependency is itself a dependent of another word, and $O(n)$ is only reached in the rare case of a cycle consisting of the entire sentence so far.

Ambiguous Dependencies Under Final Dependencies

Another bug resulted from the combination of final and ambiguous dependencies. If a dependency was final, but a dependency "under" it in the graph was ambiguous, the parser could cross the final dependency during the search for pseudo-heads, violating the projectivity requirement if an attachment was made. A sentence which exhibited this problem using JED's grammar at the time was *He has not the least concern*: as shown in Figure 4.15, the dependency between *least* and *not* was final, but the dependency between *least* and *the* was ambiguous, thus allowing the parser to skip over the former when finding dependents for *concern*. To fix this clearly erroneous behavior, another new method was introduced, named `DisambiguateBetween`, which made all dependencies final between a pair of endpoints; this would be run whenever a final dependency was created, with the ends of that dependency as its arguments.



(a) (b)

Figure 4.15 **A projectivity error created through the use of ambiguous dependencies: the parser is able to skip over *least* → *not* and reattach *the*.**

Disambiguation of Subject Modifiers

A final kluge which had to be added to the algorithm involved the fact that the attachment of modifiers and determiners must remain ambiguous. Consider the sentence *The green house paint is drying*. In this sentence, *the* is initially parsed as a dependent of *green*, then reattached to *house*, and finally reattached to *paint*. In each of these steps, it is necessary for the attachment of *the* to be ambiguous, even when it is attached to a noun.

However, this ambiguity made it impossible to reattach subordinating conjunctions to verbs in the manner preferred by Järvinen and Tapanainen when the verb had a subject with modifiers. Consider the sentence *He left after the dance ended*, shown in Figure 4.16. After attaching *dance* to *ended*, it is impossible to attach *after* as well; the parser stops searching for pseudo-heads as dependents after the attachment of *the* to *ended* fails.

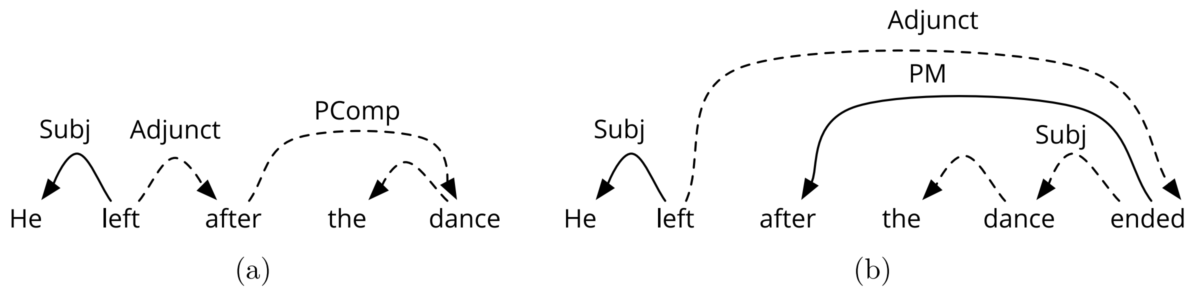


Figure 4.16 **An impossibility in *He left after the dance ended*: the correct parse is shown in (b), but it is impossible to get there from (a).**

To solve this problem, an additional statement was added immediately before the "find dependents" loop. If the previous word was a determiner, noun, or adjective, but the current word is not, the parser will make all dependencies headed by the previous word final. This is admittedly not the most elegant implementation, but it does solve the problem in parsing sentences such as that from Figure 4.16.

4.5 Modifications to Järvinen's Grammar

As mentioned previously, the grammar implemented in JED is mostly based on that of Järvinen and Tapanainen; however, there are a few notable differences which deserve further discussion, and a few structures for which no examples were shown.

4.5.1 Modifications to Existing Examples

The first difference from Järvinen and Tapanainen's grammar involves distinctions which are not necessary to DSS. The point of JED, after all, is to provide "just enough dependency" to be of use in DSS; thus, distinctions which have no effect on DSS scoring can safely be ignored. There were two main aspects of Järvinen and Tapanainen's grammar where these distinctions were truly unnecessary: adverbial adjuncts and objects.

Järvinen and Tapanainen describe a total of thirteen distinct types of adverbial adjuncts, none of which can be attached more than once to the same head. JED takes the opposite approach, also alluded to in Järvinen and Tapanainen's paper, for simplicity's sake: the grammar only includes a single type of adjunct, known appropriately as `Adjunct`, but allows an unlimited number of this type of dependent to depend on a single head.

The other distinction which was removed was between several types of complements that can follow verbs. In Järvinen and Tapanainen's grammar, these are divided into three main types: direct objects (`Obj`), datives (`Dat`), and predicative complements (`Comp`). The rules to distinguish these three types quickly grew out of hand during their development in JED; however, it was noted during the course of development that the distinction between these three types is unnecessary for the purposes of DSS, a fact which could significantly simplify the rule set. As the only distinction necessary for DSS is to distinguish subjects from all other verb complements, and as a verb can have a maximum of two following complements, the other complements were simply consolidated into a single `Obj` type.

In one case, however, a distinction was *added* in JED that was not present in Järvinen and Tapanainen's grammar, to represent a significant syntactic difference between two types of quantifiers. For an illustration of this distinction, consider the phrases *the two dogs* and *all the dogs*. Both *two* and *all* are given the same label, `Qn`, by Järvinen and Tapanainen's grammar. However, this class encompasses two grammatically distinct varieties of words: universal quantifiers, which may only precede determiners (as in *all the dogs*), and cardinal (numeric) quantifiers, which may only follow determiners (as in *the two dogs*). It was much easier to prevent unnecessary attachments by distinguishing the two types of quantifiers; in JED's grammar, these were labeled `QnUni` and `QnCd`, respectively.

The last difference from Järvinen and Tapanainen's grammar relates to the projectivity restriction. Recall, as discussed in Section 4.2.2, that Järvinen and Tapanainen's grammar is non-projective in the two constructions shown in Figure 4.5: comparatives and fronted prepositional complements. In the case of comparatives, the solution, as discussed previously, was to attach the *as/than* clause to the main adjective or adverb itself, rather than to the comparative adverb; this is shown in Figure 4.6. For fronted prepositional complements, the alternative representation that was chosen was to attach the complement to the verb itself rather than to the preposition, as shown in Figure 4.17. Although this is not as syntactically accurate as directly attaching the complement to the preposition itself, it does solve the projectivity problem while still emphasizing that the interrogative in question acts as an object.

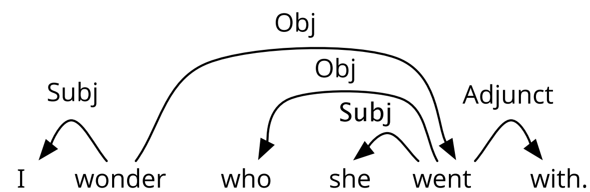


Figure 4.17 **An alternative, projective structure for *I wonder who she went with*, as implemented in JED's grammar.**

4.5.2 New Examples

A number of other constructions were significant in scoring DSS, but did not have any examples shown by Järvinen and Tapanainen. These all involved idiosyncratic constructions in which dependency relationships were not entirely clear.

The first of these constructions was *How come...?* This is generally accepted to be an abbreviated form of *How did it come to be that...*; thus, the obvious parse is to have *come* as the main verb, with an adjunct *how*, and with what follows as a subordinate clause, as shown in Figure 4.18.

Another construction omitted from Järvinen and Tapanainen's examples was *What if...?* In this case, the analysis used in JED attaches both *if* and *what* as dependents of the following verb; this is based on the analyses used by both the Stanford Parser (Klein and Manning, 2003) and MaltParser (Nivre et al., 2007b).

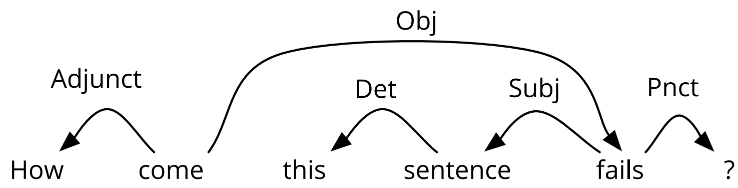


Figure 4.18 **The correct parse for *How come this sentence fails?***

Still another omitted construction involved the expression *How about... ?* This expression is often, as described in Lee (1974), followed by a gerund; for instance, *How about parsing this sentence?* To make matters worse, the two other parsers I tested gave wildly different analyses of this structure, as shown in Figure 4.19. The Stanford Parser attached both *how* and *about* as dependents of what followed; Malt, on the other hand, treated *how* as the head, *about* as a dependent of *how*, and the following gerund or noun phrase as a dependent of *about*. Malt's approach made more sense to me, as it is in line with other parses of *about*; this is the approach that was implemented in JED as well.

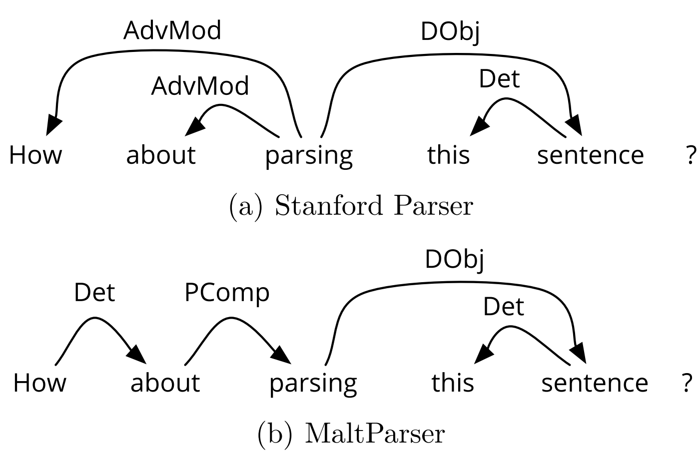
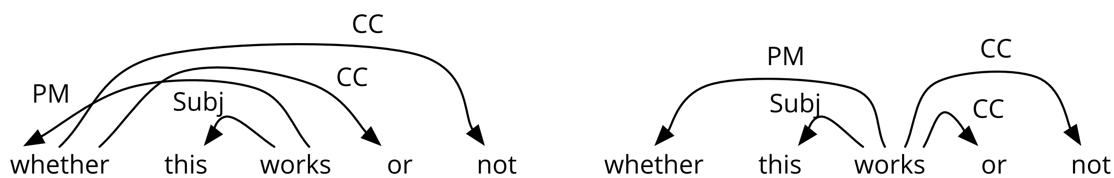


Figure 4.19 **Two competing parsers' analyses of *How about parsing this sentence?* The MaltParser approach was also followed by JED.**

The final construction involved the phrase *whether... or not*. The problem in this case is that *whether* and *or* can be divided by a clause; as shown in Figure 4.20(a), an accurate analysis of *I don't know whether to go or not* would require overlapping dependencies because of this separation. The solution which was implemented in JED, as shown in Figure 4.20(b), was to attach *or not* to the verb, as would be done in a gapping construction, when *whether* has an attached infinitive. In cases where *whether or not* was contiguous, the conjunction would be analyzed normally.



(a) Non-projective parse

(b) Projective parse

Figure 4.20 **Two parses of "whether to go or not". The first violates projectivity; the second, while less ideal, is projective.**

During the development of JED, I also discovered and corrected a significant error in my own morphological analyzer (Boisclair, 2008), which was directly inherited from the source that inspired it (Covington, 1994). Both my implementation and Covington's added an unnecessary *e* to the stem in verbs such as *seemed*, because of a logical error in the rule used to handle words such as *liked*; the latter rule should only apply when the consonant is preceded by a single vowel.

4.6 Final Rule Set

A summary of all of the rules used in the final version of JED can be found in Table 4.1.

Table 4.1: Dependency Rules from JED

Dependency	Direction	Type	Final	Notes
$\texttt{VB}x \rightarrow x$	head-first	`Obj`	false	Only if the head is a speech verb and the dependent is one quote level greater than the head.
$x \rightarrow \texttt{CC}$	head-first	`CC`	false	x cannot be tagged `CC` or `,`. If the head, or any head thereof, is a verb, mark the head's position as true in `verbCC`. Change any `UnkComma` dependents of the head to `CC`. If the head is itself attached as an `UnkComma`, change the `UnkComma` dependents of *its* head as well.
$x \rightarrow$,	head-first	`UnkComma`	see notes	x cannot be tagged `CC` or `,`. If the head, or any head thereof, is a verb, mark the head position as true in `verbCC`. If the head's head is a speech verb whose subject has a comma dependent, mark the head's position as true in `parenSaid`, delete the subject of that verb, and if it has an object, reattach that as a subject. Final if the head is a modifier of something that also has a comma dependent; ambiguous otherwise.
$x \rightarrow y$	head-first	see notes	see notes	x and y must be of compatible types, x must be at a lesser or equal quotation level to y, and x must have a dependent either of type `CC` or `UnkComma` at some level. Type is either `CC` or `UnkComma` depending on which dependent x already has. If the proposed type is `UnkComma`, the head and dependent are verbs, and the dependent has a quotation mark attached, attach as `Obj` instead.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
				If the head is *whether*, the dependent is *not*, and this is a known coordination structure, make this attachment final; otherwise, mark it as ambiguous. If the CC or UnkComma dependent of x is not a direct dependent, reattach it directly to x. Convert to a gapping relationship if x already has a non-conjunction word attached via UnkComma or CC—that is, attach the conjunction to the verb, and its former sibling to the conjunction.
NNx → NNx	head-first	Adjunct	false	Both the head and dependent must be temporal nouns.
NNx → VBN	dep-first	Attr	true	Dependent must not have a pseudo-head, must not be the head of a verb chain, and must not have a subject or prepositioned marker.
POS → NNx	dep-first	Attr	true	Head must not have any other dependents.
DT* → RB	dep-first	Adjunct	true	Adverb must be one of those shown in section 5.11 of Huddleston et al. (2002).
NNx → DT*	dep-first	see notes	false	Dependent cannot have another determiner attached anywhere or a preposition attached afterward. If the determiner is *all* or *both*, attach as QnUni. If the dependent is a cardinal number and the head has no Det attached, attach as QnCd. If the dependent is any other determiner and the head has no dependents of type Det or QnUni or tagged PRP$, WP or POS, attach as Det.
NNx → PDT	dep-first	Det	false	Head must already have a Det dependent attached.
DT* → PDT	dep-first	Det	false	Head must not already be attached as an ambiguous Det of something else.
NNx → PRP$/WP$/POS	dep-first	Attr	false	Head must not have a dependent of type Det or tagged PRP$, WP$ or POS.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
`NN`x → `PRP`	dep-first	`Attr`	false	Head must not have a dependent of type `Det` or tagged `PRP$`, `WP$`, `PRP` or `POS`. This rule exists to handle a non-standard dialect construction.
`NN`x → `JJ`x	dep-first	`Attr`	false	Head must not have a dependent of type `Det`, `QnUni` or `QnCd`. Dependent must not have a dependent of its own following it with tag `IN`.
`NN`x → `JJ`x	head-first	`Mod`	false	Dependent must either be tagged `JJR` or have its own dependent of type `Ad`.
`JJ`x → `(W)RB`	dep-first	`Adjunct`	see notes	Do not attach if the head has a dependent of type `Det`, or the dependent has an untyped dependent that follows it. If the dependent could be a subordinating conjunction, make ambiguous; otherwise, make final.
`RB`x → `(W)RB`	dep-first	`Adjunct`	see notes	Do not attach if the head has a dependent of type `Det`, or the dependent has an untyped dependent that follows it. If the dependent could be a subordinating conjunction, make ambiguous; otherwise, make final.
`NN`x`/JJ`x`/DT` → `IN`	head-first	`Mod`	see notes	Dependent must not already be an ambiguous dependent of another word, and head must not have a `CC` dependent. If the dependent could be a subordinating conjunction, make ambiguous; otherwise, make final.
`NN`x → `NN`x	dep-first	`Attr`	true	Head must not be a temporal noun that can act as an adjunct without a dependent. Head and dependent must be directly adjacent.
certain determiners → "else"/"more"	dep-first	`Det`	true	Head and dependent must be directly adjacent. Allowed determiners are *all*, *much*, *little*, and compound determiners (every/some/any/no + body/one/thing/where).
compound determiners → `JJ`x	head-first	`Mod`	true	Head and dependent must have at most one word in between them.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
`JJ`x → `PRP$/POS`	dep-first	`Attr`	false	
`DT`* → *of*	head-first	`Mod`	true	
`JJ`x`/RB`x`/IN` → *all*	dep-first	`Adjunct`	true	Do not make this attachment if the dependent is *of*; otherwise, the previous rule will be overridden.
`JJ/RB` → *as/less/more*	dep-first	`Ad`	true	
`JJ/RB` → `RBS`	dep-first	`Ad`	true	Always allow *most* and *least* as dependents, even when not explicitly tagged as `RBS`
`JJ`x`/RBS` → `DT`*	dep-first	see notes	false	Dependent cannot have another determiner attached anywhere or a preposition attached afterward. If the determiner is *all* or *both*, attach as `QnUni`. If the dependent is a cardinal number and the head has no `Det` attached, attach as `QnCd`. If the dependent is any other determiner and the head has no dependents of type `Det` or `QnUni` or tagged `PRP$`, `WP` or `POS`, attach as `Det`.
`JJR/RBR` → *than*	head-first	`Mod`	false	
`JJ/RB` → *as*	head-first	`Mod`	false	Head must have *as* already attached as a dependent.
`JJ/RB` → *than*	head-first	`Mod`	false	Head must have *more* or *less* attached as a dependent.
`JJS/RBS` → `VB`x`/MD`	head-first	`Mod`	false	
`IN/TO` → *here/there*	head-first	`PComp`	true	Head and dependent must be directly adjacent.
more → *some/any*	dep-first	`QnUni`	true	Head and dependent must be directly adjacent.
`VB`$x^{\dagger}$ → temporal noun	dep-first	`Adjunct`	false	Depending on the word used in the tail, may or may not require the dependent to have its own dependent. This decision is based on the list given in section 8.6.3 of Huddleston et al. (2002).
`VB`x → `NN`x	head-first	`Voc`	true	Head must have a comma attached, and dependent must not have a quotation mark attached. As a side effect, the dependency between the head and the comma is also converted to a `Voc`.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
VBx/MD† $\rightarrow$ NNx/PRP/POS/ WP/DT*	head-first	see notes	see notes	Do not create a dependency if the verb already has both an object and an adjunct attached, or if it has a `WRB` or `WP` following it. If the verb is an auxiliary or modal with no `Subj` dependent, then attach as a final dependency of type `Subj`. Otherwise, attach as an ambiguous dependent of type `Obj`.
VBx/MD $\rightarrow$ NNx/PRP/POS/ WP/DT*/JJx	dep-first	see notes	see notes	Type is `Subj` by default. It will instead be `Obj` if the dependent comprises an interrogative, and the head either already has a subject or is the copula. Do not attach if the head has any `VCh` dependents, is a `VBG`, or has a `PM` dependent. Do not attach if the head already has a dependent of the same type, unless the existing dependent is a `PRP` and this dependent is not a `PRP` or an ambiguous `PComp`. Do not attach if the head is a known infinitive and the dependent is an adjective that can take an infinitive adjunct. Do not attach if the dependent is already an ambiguous dependent of something else, unless: • The parent of the dependent is a possible subordinating conjunction, is a conjunction headed by a verb, has a non-interrogative subject but no `PM` or interrogative adverb attached, or has an interrogative adverb but no subject attached. • The dependent is an interrogative or part of an `UnkComma` structure.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
				Dependency is ambiguous if the dependent comprises an interrogative, the head is an auxiliary verb, and the head does not already have a subject; it is also ambiguous if the head is a speech verb and the subject has a comma attached. Otherwise, the dependency is final.
VBx/MD $\rightarrow$ EX/*here*	see notes	Subj	true	Head must not have any other Subj dependent. Either the head must be an auxiliary or modal, or the tail must be to the left of the head.
VB$x^{\dagger}$ $\rightarrow$ JJx	head-first	Obj	false	Dependent must have a Det, Attr or QnCd dependent of its own.
VB$x^{\dagger}$ $\rightarrow$ JJx/VBN	head-first	Comp	false	If the dependent is a VBN, it must not have a subject. Head must either already have an object or else be a verb that can take a complement, as listed in section 4.5.4 of Huddleston et al. (2002).
IN/TO $\rightarrow$ NNx/PRP/POS/DT*	head-first	PComp	false	Head must not already have a PComp dependent.
IN/TO $\rightarrow$ VBx/MD	head-first	PComp	false	Verb must have an interrogative dependent at some level.
IN/TO $\rightarrow$ JJx	head-first	PComp	false	Head must not already have a PComp dependent.
IN $\rightarrow$ IN/TO	head-first	_	see notes	Head and dependent must be directly adjacent. Ambiguous if the head is a subordinating conjunction and the tail is TO; final otherwise.
VBx/MD $\rightarrow$ IN	dep-first	PM	see notes	Preposition must be a subordinating conjunction; even those that were not tagged as IN will still be considered, as long as they are not tagged VBx or NNx. Do not attach if the verb already has a PM dependent, unless the verb is an infinitive with no subject; the verb is an infinitive and the proposed dependent is *for*; the head is within a verb chain; or the head is a VBG. Do not attach if the dependent has a PComp dependent and the head has a Subj. If the verb is an auxiliary, make the dependency ambiguous; otherwise, make it final.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
				If the dependent already had a pseudo-head z that preceded it (i.e., the preposition was an adjunct or modifier), attach this dependency's head as a dependent of z as a side effect, also making it final if this dependency was final.
IN/TO $\rightarrow$ VBG	head-first	PComp	see notes	Dependent must not be part of a verb chain. If the dependent is an auxiliary verb, leave this dependency ambiguous; otherwise, make final.
VB$x^\dagger$ $\rightarrow$ IN	head-first	Adjunct	see notes	If the dependent could be a subordinating conjunction, leave ambiguous; otherwise, make final.
VB$x^\dagger$ $\rightarrow$ TO	head-first	Adjunct	false	
VB $\rightarrow$ TO	dep-first	PM	true	Also mark the verb in `isInfinitive`.
VBx $\rightarrow$ VBx/MD	dep-first	VCh	true	Do not attach if dependent has an object or complement, unless the tail also has an interrogative attached. Do not attach if the head has any non-adjunct dependents, unless one of those is a determiner which can follow an auxiliary, as specified in section 5.9.2 of Huddleston et al. (2002). In that case, reattach the determiner to its head as a determiner or quantifier. If the dependent already has a pseudo-head, make that the head of this dependency's head instead, also rendering it final if the head is not an auxiliary verb itself. If the dependent has an ambiguous subject and object among its dependents and is not the copula, swap them as a side effect.
NNx/PRP/POS/DT* $\rightarrow$ VBx/MD	head-first	Mod	see notes	Dependent must have a PM dependent at some level; if the head is a dependent of anything, the PM must not be *to*. If the head has at least one preposition (IN/TO) attached, detach the last of its prepositions as a side effect.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
				If the dependent could possibly be an auxiliary verb, leave this dependency ambiguous; otherwise, make final.
`NN`x`/PRP/POS/DT`* $\rightarrow$ `VB`x`/MD`	head-first	`Mod`	false	Dependent must have either a subject or an interrogative as a dependent; in the case of a non-interrogative subject, the head must also not have a comma attached. Quotation level of the head must be equal to or greater than the quotation level of the dependent. Do not attach if the head is a `PRP` or `POS`, the head is a dependent of a verb, and that verb is not the copula.
`VB`x`/MD` $\rightarrow$ `VBG`	dep-first	see notes	true	Dependent must not be part of a verb chain. If the head already has a subject, then attach as an adjunct (i.e., participle); otherwise, attach as a subject (i.e., gerund).
`VB`x`/MD` $\rightarrow$ `VB`x	dep-first	`Adjunct`	false	The dependent either must be a speech verb offset by commas, or must already have dependents of type `UnkComma` and `Subj`.
`VB`x $\rightarrow$ `VB`	head-first	`Obj`	see notes	Head verb must be one that can take bare infinitives, as specified in section 14.5.6.2 of Huddleston et al. (2002). As a side effect, the last `Obj` of the head, if one exists, is removed and reattached as a `Subj` of the dependent. If the dependent is a possible auxiliary verb, leave ambiguous; otherwise, make final. Mark the verb in `IsInfinitive` as a side effect.
`VB`$x^{\dagger}$ $\rightarrow$ `VBG`	head-first	`Obj`	false	Dependent must not be part of a verb chain.
`VB`x $\rightarrow$ `VB`	head-first	see notes	true	Verb must already be a known infinitive.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
				If the preceding word comprised by the head verb could be a subject of the infinitive and the head verb is one that can take a complex infinitive, reattach that word as a `Subj` of the infinitive verb, then attach the infinitive as a `Obj` of the head verb. In addition, if the preceding word had been the complement of a preposition, attach that preposition to the infinitive as a `PM`. If there is no potential subject for the infinitive and the head verb is one that can take a simple infinitive complement, attach the infinitive as an `Obj`. In all other cases, attach the infinitive as a `Adjunct`.
VBx $\rightarrow$ VBx/MD	head-first	`Obj`	see notes	Do not attach if the dependent is an ambiguous modifier of something else, if the head has a `PM` but the dependent does not, if the head comprises an interrogative and subject but the dependent does not comprise an interrogative, or if the head is a parenthetical speech verb. Leave ambiguous if the dependent verb is a `VBN` or auxiliary; otherwise, make final.
VBx $\rightarrow$ VBx/MD	dep-first	`Subj`	true	Do not attach if the dependent is an ambiguous modifier of something else, if the head has a `PM` but the dependent does not, if the head comprises an interrogative and subject but the dependent does not comprise an interrogative, or if the head is a parenthetical speech verb. Only attach in this direction if the dependent has a `PM` and the head does not, if the dependent comprises both an interrogative and subject and the head does not comprise an interrogative, or if the head is at the end of a complete verb chain and the tail is not.
JJx $\rightarrow$ VB	head-first	`Mod`	false	Dependent must be marked in `isInfinitive`.
NNx $\rightarrow$ VBG	head-first	`Mod`	true	Dependent must not be part of a verb chain.

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
NNx → VBG	dep-first	Attr	false	Dependent must not be part of a verb chain. Dependent must have a Det, QnCd, QnUni or Attr dependent and the head must have none of those.
VBx/MD† → RBx/WRB/EX	head-first	Adjunct	see notes	If the dependent is *not* or *n't*, leave ambiguous; otherwise, make final.
VBx/MD† → RBx/WRB/EX	dep-first	Adjunct	true	Do not attach if the head already has a Subj or Obj, unless the dependent is a WRB and the head is not yet part of a verb chain. Also do not attach if the dependent has an untyped dependent that follows it.
VBG → DT*	dep-first	see notes	true	Do not attach if the dependent is *we* or *you* or the head is part of a verb chain. Dependent cannot have another determiner attached anywhere or a preposition attached afterward. If the determiner is *all* or *both*, attach as QnUni. If the dependent is a cardinal number and the head has no Det attached, attach as QnCd. If the dependent is any other determiner and the head has no dependents of type Det or QnUni or tagged PRP$, WP or POS, attach as Det.
VBG → JJx	dep-first	Attr	true	Head must not be part of a verb chain.
VBG → PRP$/WP$/POS/PRP	dep-first	Attr	true	Head must not be part of a verb chain. If the tail is a PRP, it must not have any pseudo-head.
VBx/MD → PRP$/POS	dep-first	Subj	true	Dependent must not have any pseudo-head.
JJ/RB → *this*/*that*	dep-first	Adjunct	false	Head and dependent must be directly adjacent.
PRP → *all*/*both*	head-first	QnUni	false	Head and dependent must be directly adjacent.
PRP → CD	head-first	QnCd	false	Head and dependent must be directly adjacent.
VBx/MD → *no*	dep-first	Adjunct	true	This is to allow for the dialect construction, e.g., "He no walk."
VBx/MD → *what*	dep-first	_	true	Head must already have *if* as a PM dependent.
WP/WRB → *about*	head-first	_	true	Head and dependent must be immediately adjacent.
x → ‘‘/(	dep-first	_	true	

Continued on next page

Table 4.1 – continued from previous page

Dependency	Direction	Type	Final	Notes
$x \to$ ''/)	head-first	_	true	Head must not be closing punctuation or a comma.
$x \to$.	head-first	Pnct	true	Also triggers the routine to correctly attach a tag question as a side effect.
UH $\to$,	dep-first	_	true	Head and dependent must be directly adjacent. Marks the position of the dependent in interjComma as a side effect.
, $\to$ UH	dep-first	_	true	Head and dependent must be directly adjacent. Marks the position of the head in interjComma as a side effect.
$x \to$ UH/,	dep-first	_	true	If the dependent is a comma, it must be marked in interjComma. The tag of the head must not be a punctuation tag.

*This is shorthand for any word tagged DT, WDT or CD, as well as words from section 5.4 of Huddleston et al.

†Can also apply to any coordinating conjunction with a verb head; if this is detected, gapping will be introduced.

Chapter 5

The User Interface: A Brief Digression

Although the main focus of this project is to improve upon the accuracy of existing applications for automated DSS scoring, the design of the application's user interface must also be given some thought. If SYCORAX is too difficult for new users to learn, this will likely hinder its adoption in spite of the improvements in accuracy.

Thus, in this chapter, I will compare and contrast the three leading applications for automated DSS from a usability standpoint. I also present the design chosen for the interface of SYCORAX, which combines features available in all three existing applications' interfaces in an extremely user-friendly manner.

5.1 Computerized Profiling

The Computerized Profiling application (Long et al., 2006) dates back to a 1986 application developed for MS-DOS; although improvements have been made to both its algorithms and its user interface, the latest version of CP, released in 2006, still remains a 16-bit DOS application. In the years since its release, 64-bit versions of Windows have become prevalent, which by design cannot run 16-bit applications; as a result, the application will not run natively at all on some modern-day Windows systems. In order to even run CP on my own Windows system, which runs a 64-bit build of Windows 7, it was necessary to use the Windows XP virtual machine provided as an optional feature by Microsoft.

As is visible from Figure 5.1, the interface is clearly dated; nonetheless, for a DOS application, it is reasonably user-friendly, driven by menus rather than a command line. However, for automatically generating a DSS analysis from a transcript, the interface is extremely

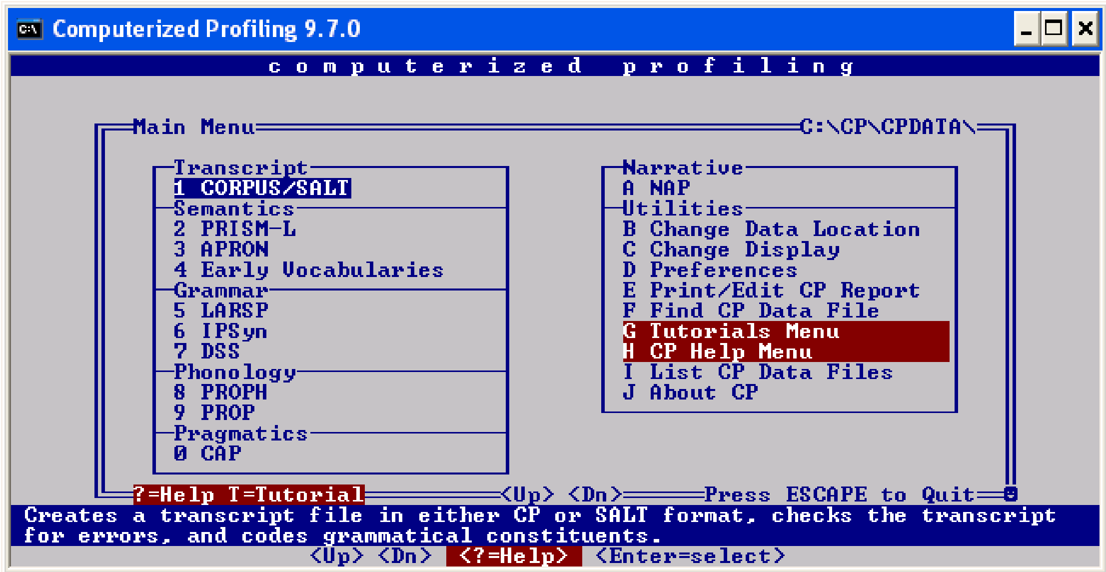


Figure 5.1 **The main menu screen from Computerized Profiling.**

cumbersome. A user must navigate through two separate menus (Figures 5.2–5.3) to create a tagged CORPUS file from an existing transcript, another menu (Figure 5.4) to generate a LARSP analysis, and yet another (Figure 5.5) to produce a DSS score from the LARSP analysis. In the case of DSS, the user must also specify a number of options (Figure 5.6) for each run. All of these steps must be taken for *each* transcript that needs to be analyzed; there is no batch capability to process multiple files at once, nor is there any way to run all the steps of a single analysis as a batch.

5.2 CLAN

Unlike Computerized Profiling, CLAN is written using modern GUI libraries, and can run natively under modern versions of Windows and Mac OS. Despite this, however, it is ironically *less* user-friendly than CP.

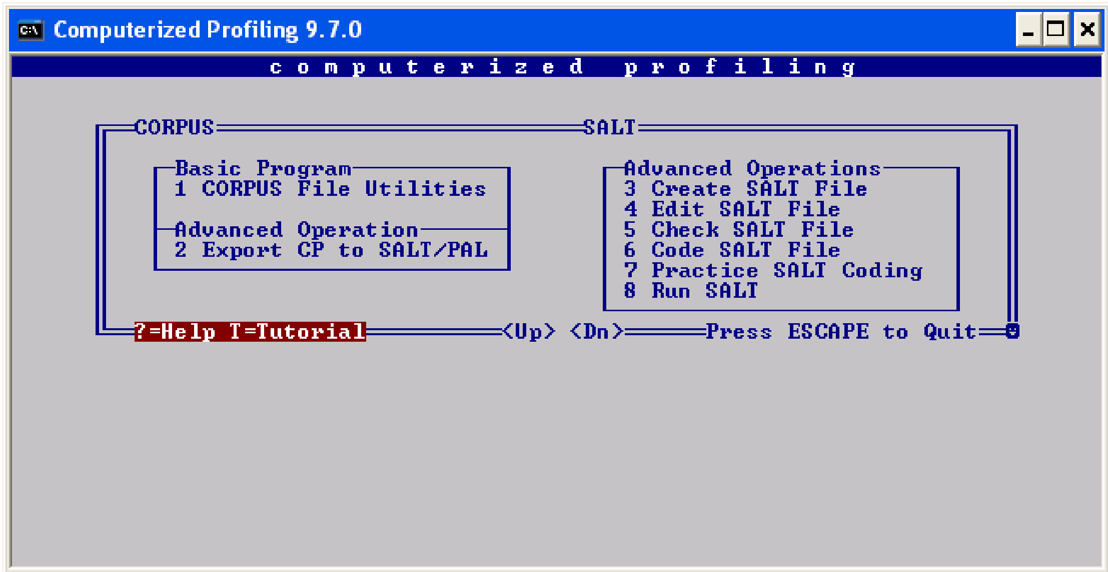


Figure 5.2 **The first CORPUS menu from Computerized Profiling.**

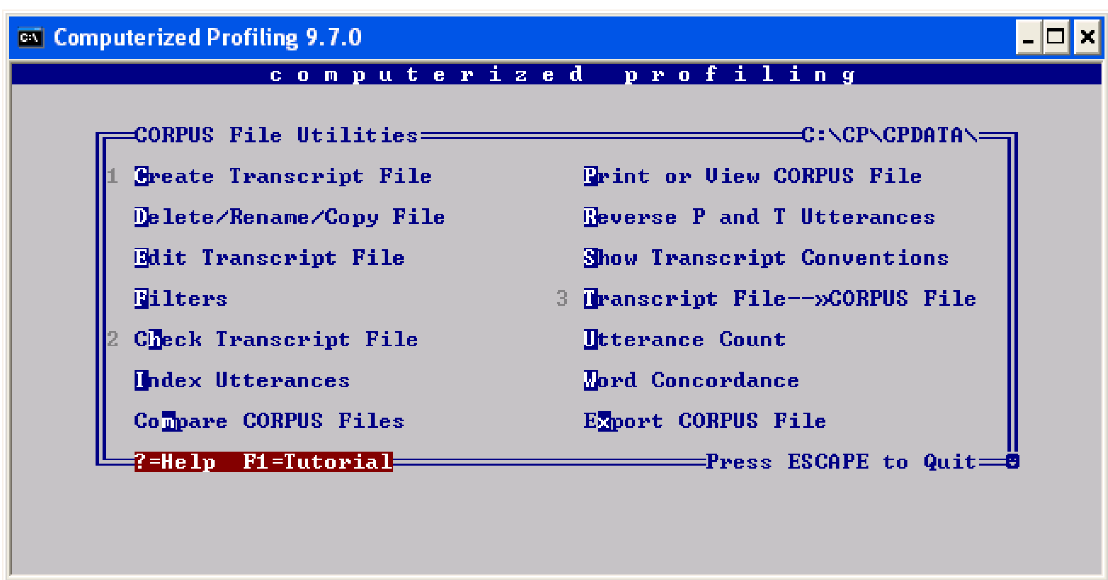


Figure 5.3 **The second CORPUS menu from Computerized Profiling.**

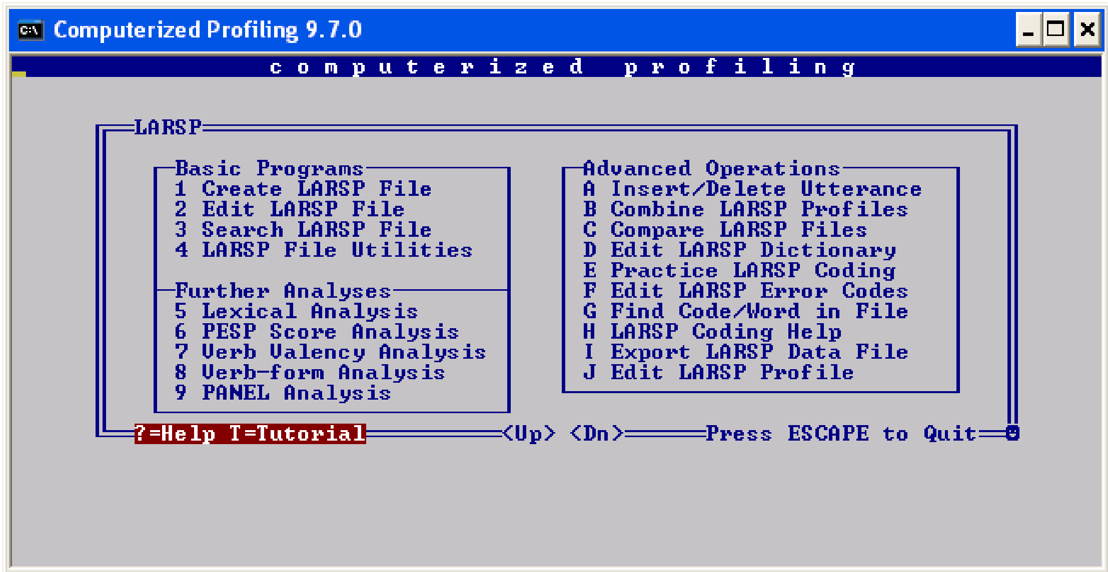


Figure 5.4 **The LARSP menu from Computerized Profiling.**

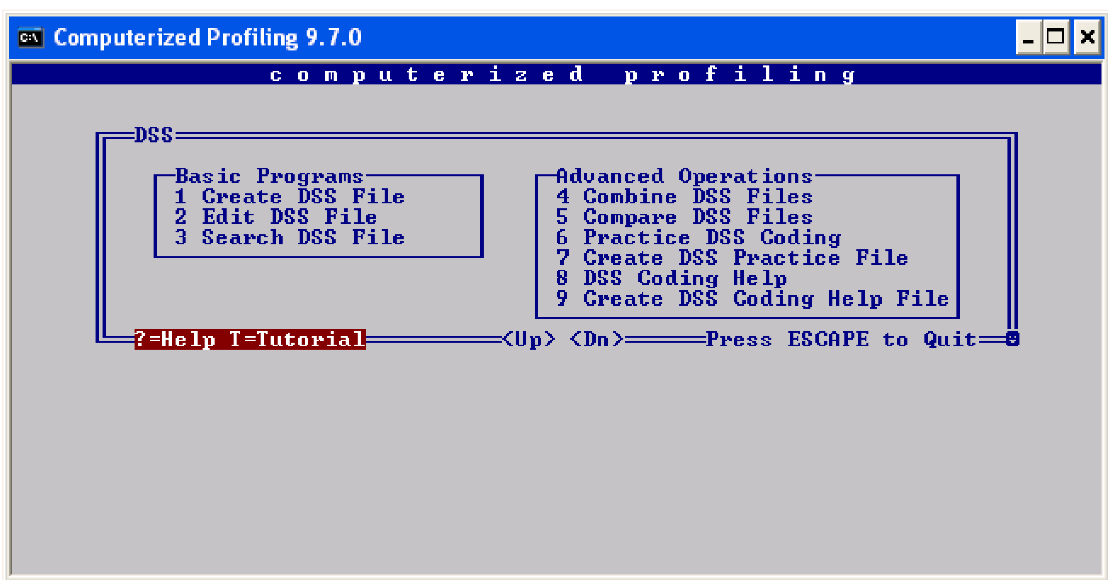


Figure 5.5 **The DSS menu from Computerized Profiling.**

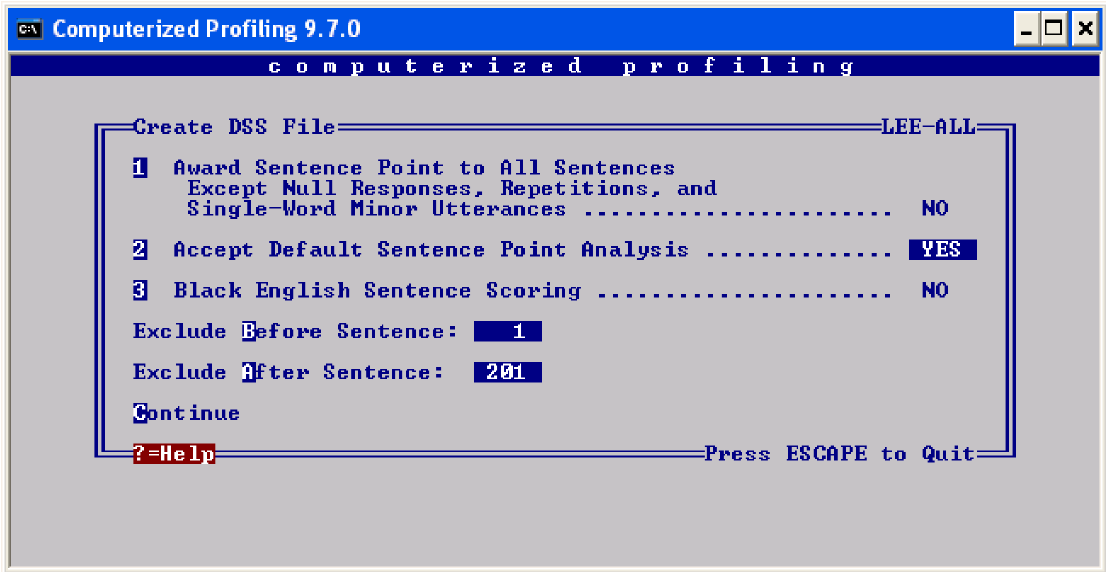


Figure 5.6 **The DSS option screen from Computerized Profiling.**

The CLAN user interface consists of two windows: an output window (Figure 5.7), showing messages generated during the analysis procedure, and an input window (Figure 5.8), in which commands are entered by the user.

As is obvious from these screenshots, rather than being entirely menu-driven like CP, CLAN is essentially a set of command-line programs encapsulated within a GUI application. This design does make CLAN more suited to batch analyses than CP, as it is possible to create a batch script to run a full analysis on a collection of files. However, the syntax of the commands, though fully documented in the application's manual (MacWhinney, 2000), is extremely arcane, and no help is offered for novice users through the GUI. Note, for instance, the command entered in the screenshot in Figure 5.8: `dss +bCHI +c200 +e +le lee-all.mor.cex`. This is the command to run a fully automated DSS analysis on the entirety of the `lee-all` file after a morphological analysis has already been run:

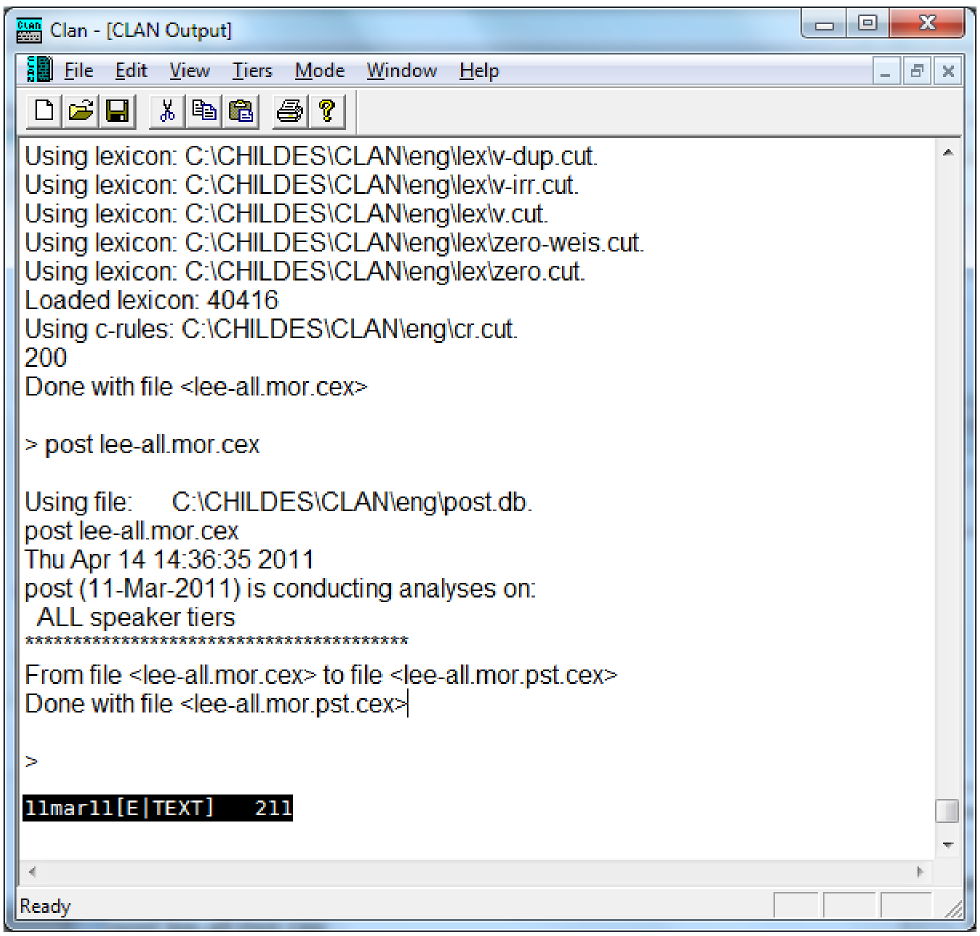


Figure 5.7 **CLAN's output window.**

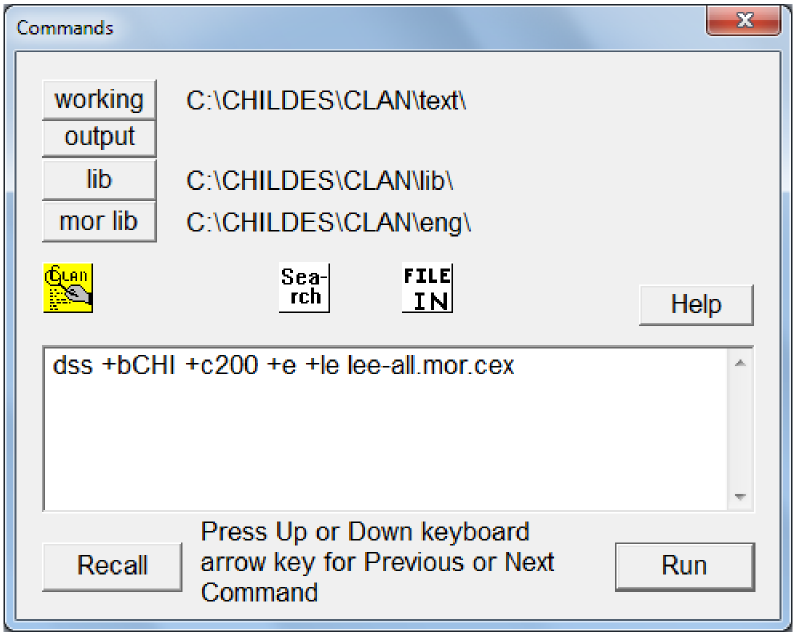


Figure 5.8 **CLAN's input window.**

- **`dss`**: The command to be run; in this case, a DSS analysis.
- **`+bCHI`**: An option to identify which speaker's words to analyze in the transcript. In this case, the speaker labeled as `CHI` is the one under investigation.
- **`+c200`**: Analyze all 200 sentences spoken by `CHI`, not just the first fifty.
- **`+e`**: Run in fully automated mode.
- **`+le`**: Use the rules for English rather than Japanese.
- **`lee-all.mor.cex`**: The name of the file generated by the morphological analysis step.

Much like CP's antiquated interface, this user-unfriendly design is largely an artifact of the application's age. Pye (1994) criticized the "cumbersome command lines" in a much earlier version of the application, and urged the developers to create a more user-friendly interface; unfortunately, seventeen years after Pye's review, no significant progress has been made in response to that criticism.

CLAN is an oddity in another way, also criticized by Pye: because it is designed for such a wide variety of linguistic analyses, its expected transcript format is a non-standard one invented by MacWhinney, known as CHAT. This format adds a variety of new codes for various linguistic phenomena, as well as the ability to track multiple speakers in a single conversation, but in doing so, it also disregards prior practices for transcribing language samples. As Pye observes, the most minimal version of CHAT is mostly compatible with the de facto standard SALT format for transcripts, and it is possible to convert between the two formats—but nowhere in the documentation does MacWhinney lay out the differences between CHAT and SALT (e.g., speaker identifications) or give any suggestions on reformatting transcripts.

5.3 DSSA

Channell's post-CP project, DSSA, takes a completely different approach with its GUI. Unlike both CP and CLAN, which automate a variety of linguistic analyses, DSSA is designed specifically with the purpose of calculating a DSS score directly from a transcript. Thus, its interface, shown in Figure 5.9, is as simplistic as possible; it is a native Mac OS X application whose main window consists entirely of an explanation of the program's expected format and a single button to perform an analysis.

Clicking that button opens a standard Mac OS dialog box to select a file to analyze, then a second dialog box to enter a file name for output. The analysis is then performed, with the full table written to the specified output file; upon completion, the final score is displayed in a dialog box, as shown in Figure 5.10.

Unlike the two competing applications, DSSA offers no options whatsoever regarding details of the DSS analysis. Most notably, the program does not even provide an option as to whether the entire transcript, or just the first fifty sentences, is to be analyzed.

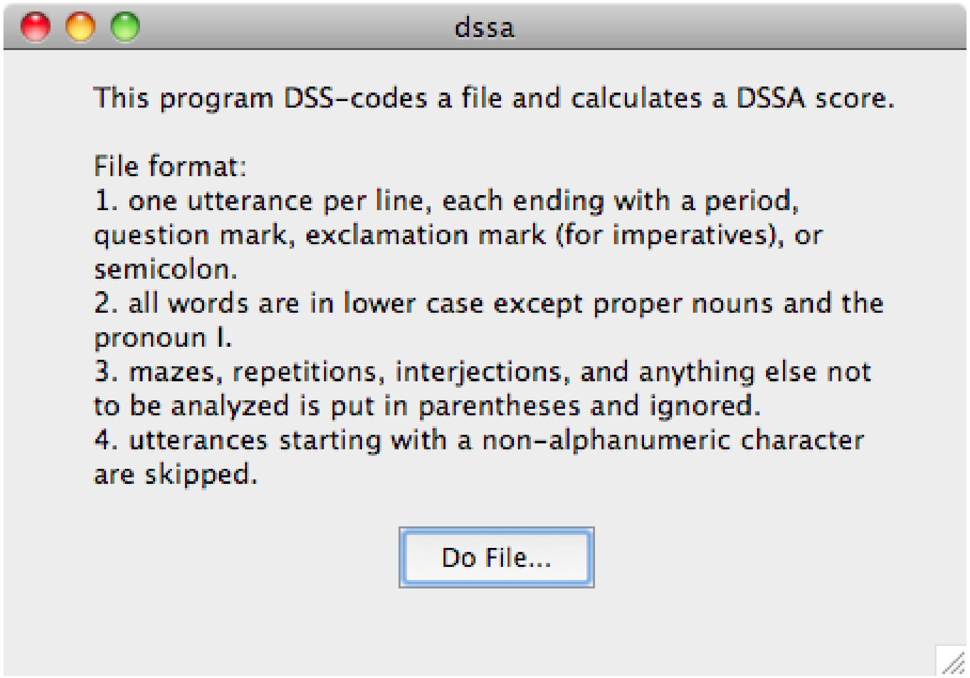


Figure 5.9 **DSSA's main window.**

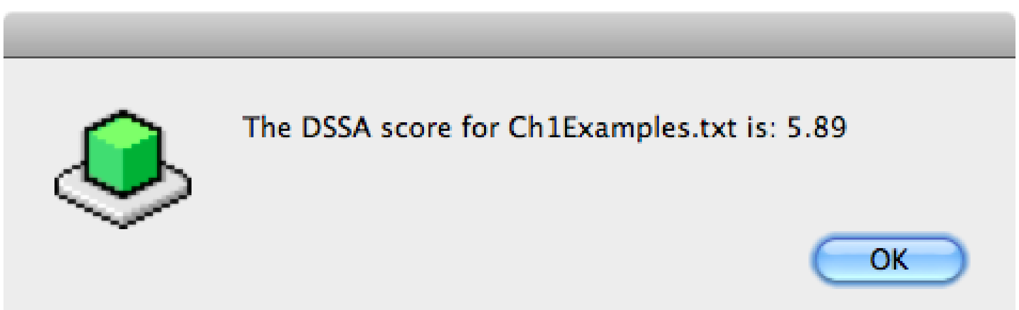


Figure 5.10 **The dialog box displayed by DSSA upon completion.**

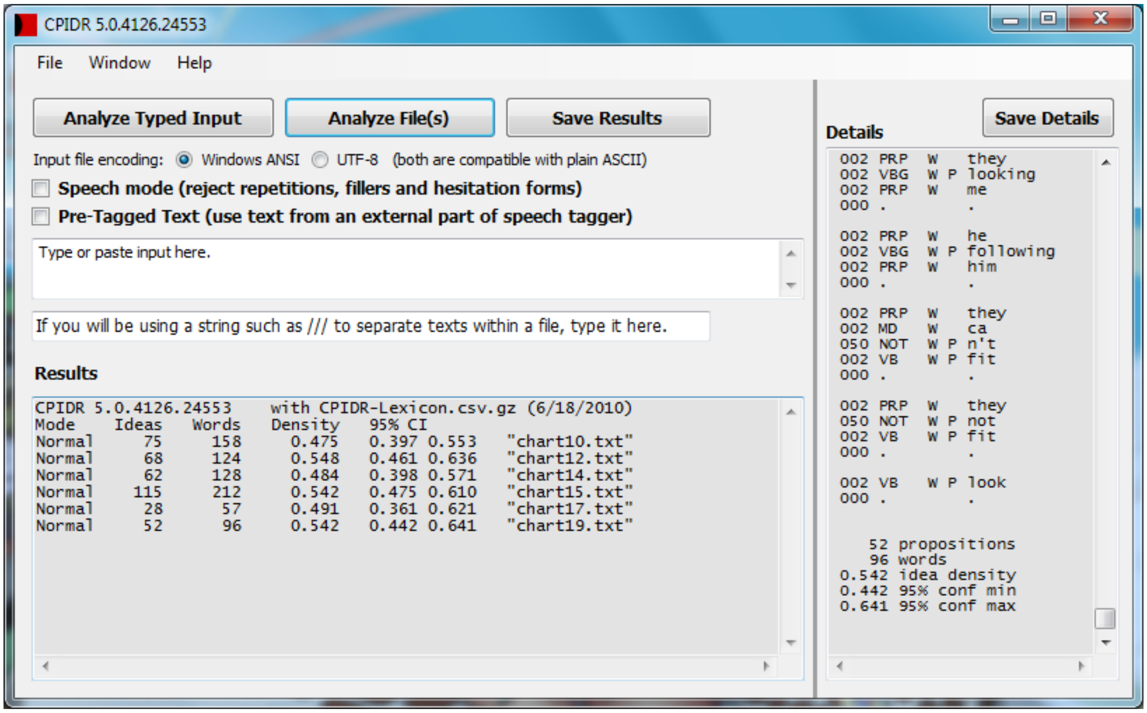


Figure 5.11 **The user interface of CPIDR version 5.0.**

5.4 SYCORAX

As SYCORAX was designed to be a syntactic analogue of CPIDR (Brown et al., 2008), so too was its user interface designed to closely parallel that of CPIDR. For the sake of comparison, the user interface of version 5.0 of CPIDR is shown in Figure 5.11, while that of SYCORAX is shown in Figure 5.12.

The interface in SYCORAX is divided into three main sections:

- **Input and options.** Here, the user can input a string to be analyzed, open a dialog box to select one or more text files to be analyzed, or set options regarding the analysis.

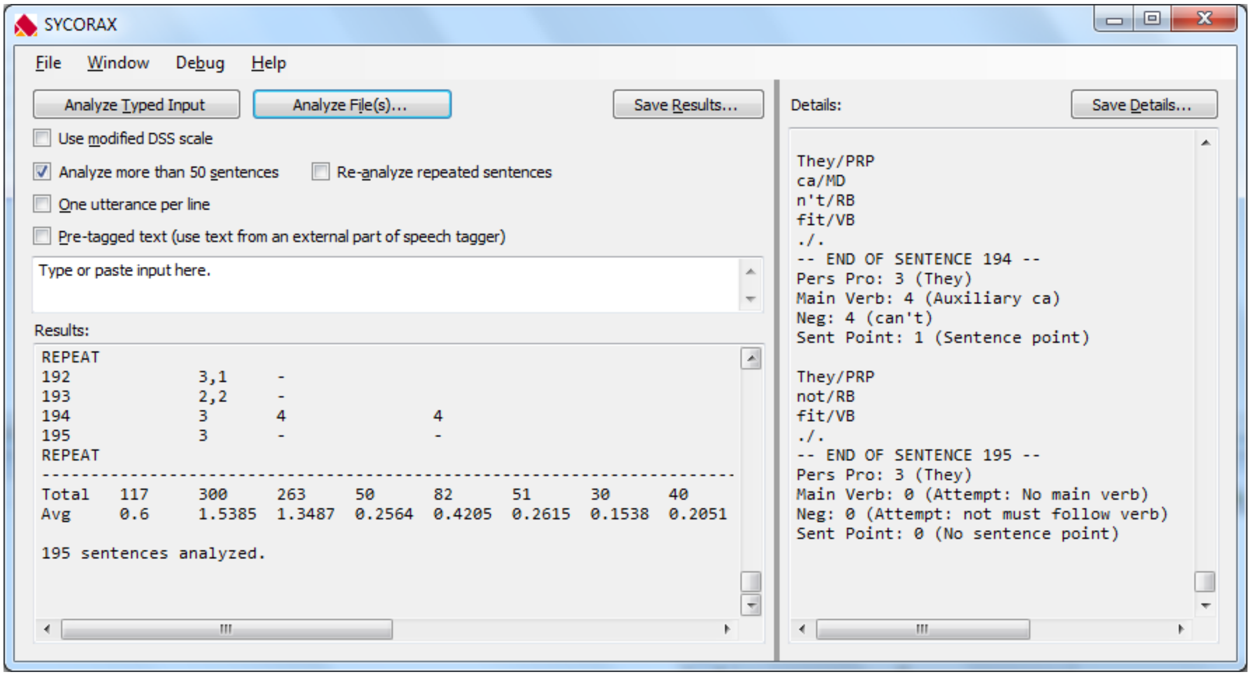


Figure 5.12 **The user interface of SYCORAX.**

- **Results.** The actual DSS table is output here for each input file or string, in a form that is not only human-readable but also tab-delimited in order to simplify computerized analysis of the data.

- **Details.** As the name suggests, this displays a more detailed version of the results, indicating not only the scores in each column but also what structures contributed those scores. In addition, the part-of-speech tags assigned to each token in the text are shown here.

To analyze any number of files at once, the user simply clicks on the **Analyze File(s)...** button at the top of the window. This opens a dialog box allowing the user to select one or more text files. Upon clicking **OK** in this dialog box, each file is given a DSS analysis, and the resulting scores are output in the result and detail areas. It is then possible to save the results or details to a text file using the corresponding **Save** buttons at the top of the window. Note that the text boxes are not cleared unless the user specifically chooses to, using a command in the **Window** menu; thus, it is possible to save all results or details from a batch as a single file.

For further troubleshooting, it is also possible to open an additional window which shows a representation of the parse tree generated for each sentence, by selecting the **Show Parse Results** command from the **Debug** menu. This window is shown in Figure 5.13.

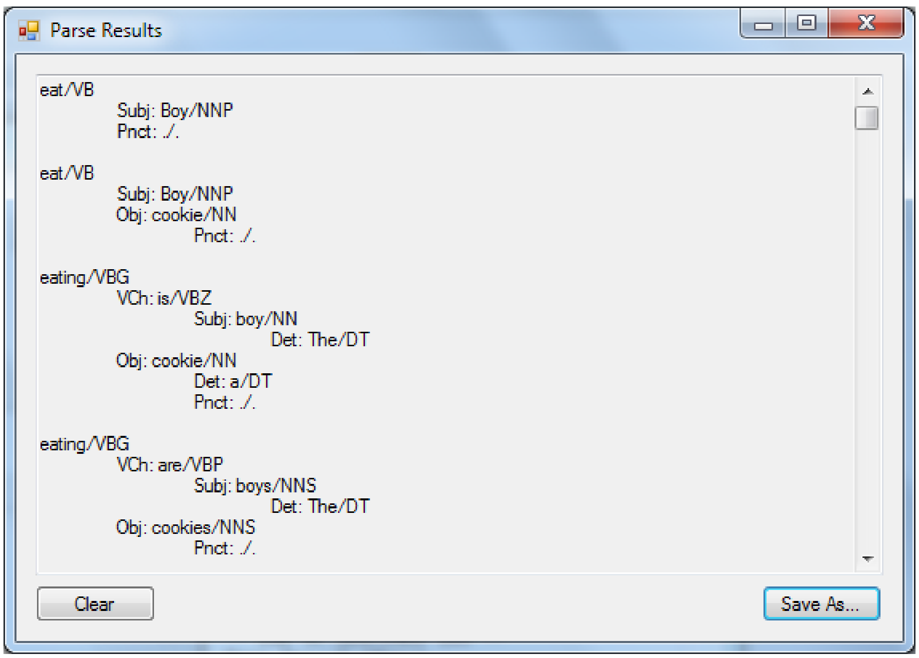


Figure 5.13 **The parse tree display of SYCORAX.**

Note that, like CP and CLAN and unlike DSSA, there are a number of options that can be set before analyzing a transcript. Some of these allow transcripts to be in nonstandard formats, including processing by an external part-of-speech tagger; others modify the scoring method with several variations on the published DSS standard, as discussed in Chapter 2. These options are defined as follows:

1. **Use modified DSS scale:** If checked, this causes certain rules to deviate from those described by Lee (1974) in the manner described in Chapter 2 of this dissertation.

2. **Analyze more than 50 sentences:** If checked, SYCORAX will analyze all sentences in the input; if not, SYCORAX will only analyze the first fifty sentences in each transcript, as originally specified by Lee.
3. **Re-analyze repeated sentences:** If checked, SYCORAX will include verbatim repetitions of prior sentences when calculating the DSS score, contrary to Lee's transcription guidelines. If unchecked, SYCORAX will ignore repetitions if they appear in the transcript, as all prior DSS applications have done.
4. **One utterance per line:** If checked, SYCORAX will consider each line of the input to be a separate utterance, regardless of punctuation. If unchecked, SYCORAX will instead attempt to intelligently find the ends of utterances based on end-of-sentence punctuation, while ignoring line breaks entirely.
5. **Pre-tagged text:** If checked, SYCORAX will interpret the input as a series of tagged tokens of the form "`token/tag`." If unchecked, it is assumed to be untagged, and SYCORAX will use its built-in tokenizer and tagger to process the text.

Note that when all options are unchecked, DSS analysis will be performed following the guidelines laid out by Lee (1974). This was an intentional decision; by default, SYCORAX matches the default behavior of prior automated applications, with variations in scoring only applied when activated explicitly by the user.

Chapter 6

A New DSS Analyzer

Now that the parser had been developed and a user interface had been designed for SYCORAX, it was time to incorporate the parser into SYCORAX's DSS rules. As hypothesized in Chapter 1, the addition of a parser and the creation of a suitable set of parsing-based rules should allow SYCORAX to exceed the accuracy of existing automated DSS applications. At the same time, with JED used instead of an exhaustive parser, performance should still be comparable to that of existing DSS analyzers.

Indeed, both of these hypotheses were shown to be true through tests on a variety of manually-scored transcripts, some even designed specifically to test constructions which are often scored incorrectly by human raters. After its DSS rules had been optimized sufficiently to use the output of the parser, the scores generated by SYCORAX were found to be more accurate, with respect to both point-by-point agreement and correlation with manual scores, than those of Computerized Profiling, CLAN, and DSSA. With additional minor tweaks to the construction of its rules, SYCORAX even surpassed the accuracy of DSSA on the two large real-world corpora used by Judson (2006).

The performance claims regarding the JED parser were also upheld during this experiment. Although SYCORAX was more accurate than its competition, its execution time was no worse than that of its competition, and in fact was sometimes better. Memory usage did exceed that of its competition, but did not even approach that of the most memory-efficient MaltParser model, indicating that JED was significantly better optimized for efficiency.

6.1 Initial Testing

For initial testing of SYCORAX, six sample transcripts from Lee (1974) were used; these samples are given in Charts 10, 12, 14, 15, 17 and 19 of Lee's book, along with manually calculated scores for each sentence, and are reproduced in Appendix C of this dissertation. Chart 10 shows a hypothetical corpus of thirty sentences that exhibit a variety of structures scored in DSS, while the other five charts are derived from actual interviews of children at various stages of language development. The six transcripts comprise a total of 201 sentences; however, four of these sentences (*I don't know* two times, *Look* two times, *I know,* and *They fall*) are repeated across multiple transcripts, giving 195 unique utterances.

To emulate how a raw transcript would be scored by a human rater, all tests were performed with input sentences transcribed exactly as spoken. This means that all parenthetical notes regarding the intended meanings of words and omitted words would not be entered in the transcript as given to the program. In addition, all false starts and sentence-initial conjunctions were omitted, as neither of those is to be counted in scoring the sentence according to Lee's rules.

Tests were run on the same set of texts using SYCORAX, Computerized Profiling and CLAN. For consistency with prior experiments, a set of agreed-upon options were used in the latter two applications. Computerized Profiling was run using the same parameters specified in Channell (2003): always accept the computer's analysis of *'s* in the creation of the corpus file; answer **yes** to both questions asked by CP regarding the LARSP (Language Assessment, Remediation and Screening Procedure) analysis; and answer **no** to question 1 and **yes** to question 2 in the DSS analysis. CLAN was run with the parameters given for automated analysis in the CLAN manual (MacWhinney, 2000): `MOR +c`, no arguments for `POST`, and `DSS +le +e`.

Accuracy was measured using the same two metrics used by Channell (2003), as discussed in Chapter 1: point-by-point agreement and the Pearson correlation coefficient. The latter is the standard statistical measure, calculated as:

$$\frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

where n is the total number of sentences, X_i and Y_i are the manual and automated scores for sentence i, and $\bar{X}$ and $\bar{Y}$ are the respective mean scores.

To calculate point-by-point agreement, the DSS table for each sentence must first be transformed into a series of codes; each score within a column is written using a particular code, with a score of zero (e.g., incomplete marks and attempt marks) represented by omitting the code entirely. For each sentence, the lists of codes are compared. If a code exists in both the manual and automated score, it is considered an *agreement*; if it exists in the manual score but not the automated score (i.e., a false negative), it is a *miss*; and if it exists in the automated score but not the manual score (i.e., a false positive), it is an *intrusion*. Each code is then marked after comparison so that multiple occurrences of the same code are counted separately. The total point by point agreement for a sentence can then be calculated as:

$$\frac{agreements}{agreements + misses + intrusions}$$

Agreement for an entire transcript can be found by summing the values of agreements, misses and intrusions over all sentences, and performing the above calculation on these totals.

Correlation coefficient and point-by-point agreement are complementary metrics of accuracy; each measures something that the other does not, and thus both are worth analyzing. The correlation coefficient indicates how closely the overall scores produced by an automated DSS analyzer approximate manual scores on the same text. Point-by-point agreement, in contrast, indicates how often the individual structures in DSS are analyzed correctly on average. This is an important distinction; an automated DSS analyzer may get reasonably accurate overall scores for entirely wrong reasons, or alternatively, may drastically diverge from human scores on one particular structure alone.

Finally, there was a question of how to deal with the above-mentioned repeated sentences; although these would not be an issue in determining the accuracy of each individual transcript, they did pose a problem in finding the accuracy of the entire corpus. As a compromise, each analysis was run on the whole corpus twice: once ignoring all but the first occurrence of each sentence, as was done in prior experiments such as Channell's, and once counting each occurrence of the repeated sentence separately. The sentence *They fall* was counted twice in the former case; each occurrence had been given a different score by hand, and so ignoring either occurrence would be an omission.

6.1.1 Before Parsing

Before JED had been incorporated, SYCORAX initially used a simpler scoring algorithm based on local context, not unlike the existing applications for automated DSS. An initial set of tests was run using this preliminary version of SYCORAX as a baseline measure to determine how well it performed in comparison to its competition.

For this experiment, one particular formatting adjustment was made in the input given to CLAN and CP. The transcription standard used by both of these applications requires that imperative sentences be indicated with an exclamation point; unlike SYCORAX, there is no heuristic implemented to guess when a sentence is imperative. The DSS transcription standards, on the other hand, require no such marking; many imperatives are shown ending in a period in the example sentences.

With this modification made to the input, experiments were run to find point-by-point accuracy for all three programs on each transcript and correlations on the entire data set. The results of these experiments are shown in Table 6.1. For the sake of comparison, the same experiment was then run without explicit marking of imperatives. The results are shown in Table 6.2; as expected, CLAN and CP both performed slightly worse, while SYCORAX had no change in accuracy.

Table 6.1 **Accuracy of automated DSS tools on sample texts from Lee (1974), with explicit marking of imperatives.**

	Transcript	**SYCORAX**	**CP**	**CLAN**	**CLAN No Post**	**CLAN '09**
Point-by-Point Agreement	10	96.9231	78.7097	20.7447	20.8738	49.6454
	12	50	42.4528	38.4615	35.3535	55.1724
	14	58.3333	46.9027	30.8511	28.8288	40.7407
	15	78.1726	87.3016	27.8846	29.8387	46.1957
	17	74.1935	42	29.0323	24	46.875
	19	70.6522	63.3663	30	34.2857	40.7407
	All	73.817	65.8263	27.8665	28.3272	46.4471
	All Unique	73.1826	65.6652	28.0597	28.25	46.7857
Correlation	All Unique	0.951147	0.931905	0.740632	0.597154	0.709077

Table 6.2 **Accuracy of automated DSS tools on sample texts from Lee (1974), with no explicit marking of imperatives.**

	Transcript	**SYCORAX**	**CP**	**CLAN**	**CLAN No Post**	**CLAN '09**
Point-by-Point Agreement	10	96.9231	78.7097	20.4301	20.6897	46.8085
	12	50	42.4528	38.4615	35.3535	55.1724
	14	58.3333	46.9027	28.7234	27.027	37.037
	15	78.1726	86.2434	27.4038	29.8387	45.6522
	17	74.1935	42	29.0323	24	43.75
	19	70.6522	58.4158	26.9663	32.6923	33.3333
	All	73.817	64.8459	26.9679	27.8528	43.8475
	All Unique	73.1826	64.6638	27.1364	27.7638	44.4643
Correlation	All Unique	0.951147	0.929817	0.731800	0.586198	0.701165

This was already a good start; on the whole, this preliminary version of SYCORAX performed more accurately, with respect to both point-by-point agreement and correlation, than CP and several versions of CLAN. However, although SYCORAX performed better than CP on most of the transcripts from Lee, it performed almost 9% worse on Chart 15, spoken by the most advanced speaker of Lee's examples. Furthermore, although SYCORAX's correlation coefficient was well above the 95% threshold considered excellent by Long and Channell (2001), the point-by-point agreement still remained below the 85% threshold considered acceptable.

Notes on CLAN

A few additional notes are necessary regarding the results from CLAN, which reveal a number of unexpected patterns.

The results labeled as "CLAN" in Tables 6.1–6.2 were generated using the March 11, 2011 build of CLAN. Versions released between June 2009 and March 2011 included a logic error which prevented any sentence with a noun rather than a pronoun as its subject from being analyzed. This error was reported to the CLAN developers in March 2011, and a fix was specifically released in response to this bug report.

One interesting aspect of CLAN is that sentences are initially tagged with multiple possible parts of speech by the morphological analyzer (`MOR`); these are then disambiguated in a second step (`POST`) before the DSS analysis is performed. To determine how much of an effect this `POST` step had on the DSS results, a second trial was performed in which the `MOR` output was fed directly to the DSS analyzer; this is shown as "CLAN No Post" in Tables 6.1–6.2. Oddly, although the correlation coefficient was significantly lower without `POST` than with it, the point-by-point agreement was slightly higher without `POST`.

For comparison's sake, the same sentences were also run through an older version of CLAN released in May of 2009, before this bug was introduced; this is shown as "CLAN '09" in Tables 6.1–6.2. This earlier version of CLAN had not yet incorporated the `POST` algorithm, and its DSS rule set was therefore significantly more complex. Although the 2011 version of CLAN produced a slightly higher correlation coefficient than the 2009 version, the latter produced over 18% higher point-by-point agreement than the former on the Lee samples.

6.1.2 Incorporating the Parser

It was now time to incorporate the JED parser into SYCORAX. This required revising many of the DSS rules to take advantage of the details available in the parse tree; as with the modifications to the tagger and parser, an iterative approach was taken in which the rules were improved in several distinct stages. The development process is summarized in Table 6.3.

Table 6.3 **Accuracy of automated SYCORAX DSS analysis at various stages of parser integration.**

	Version	**10**	**12**	**14**	**15**	**17**	**19**	**All**	**All Unique**	**Correl.**
0	No Parse	96.9231	50.0000	58.3333	78.1726	74.1935	70.6522	73.8170	73.1826	0.951147
1	Remove Pre-Proc	89.4737	50.0000	58.3333	77.2727	70.9677	70.6522	71.9436	71.2681	0.913953
2	Reversal, QTag	90.2256	50.0000	58.3333	77.7778	70.9677	70.6522	72.2571	71.5891	0.918414
3	Verbs	94.6970	62.5000	66.6667	87.2222	88.4615	82.5000	82.2300	81.7531	0.946828
4	Rev/Neg Errors	96.1538	66.1765	68.2927	87.2222	88.4615	82.5000	83.3922	82.9401	0.948849
5	Improper Verbs	97.6563	64.7059	73.6842	89.3258	88.4615	84.6154	85.3791	84.9722	0.961681
6	Indef/Mod	97.6563	67.6923	73.6842	89.8305	88.4615	85.7143	86.1566	85.7678	0.971323
7	Pers. Pronouns	97.6563	68.7500	74.6667	90.8571	82.1429	87.8378	86.7647	86.3894	0.972273
8	More Pronouns	97.6563	69.8413	74.6667	91.4286	88.4615	87.8378	87.4307	87.0722	0.974237
9	Negatives	98.4252	69.8413	74.6667	91.4286	88.4615	89.0411	87.7551	87.4046	0.978602
10	*Wh*-Questions	98.4252	69.8413	74.6667	91.9540	88.4615	89.0411	87.9182	87.5717	0.978642
11	Conjunctions	98.4252	69.8413	74.6667	92.4855	88.4615	89.0411	88.0819	87.7395	0.978418
12	"Whether or not"	98.4252	69.8413	74.6667	92.4855	88.4615	89.0411	88.0819	87.7395	0.978418
13	QTag fixes	98.4252	69.8413	74.6667	92.4855	88.4615	89.0411	88.0819	87.7395	0.978418
14	Misc. fixes	98.4252	69.8413	74.6667	92.4855	88.4615	89.0411	88.0819	87.7395	0.978418

The most significant improvement resulted from a series of modifications involving the rules for the two verb scores (main and secondary verbs). In this case, the greatest problem was that question inversion was no longer handled through transformations that un-inverted the sentence, and so rules which had been based on the transformed sentence were rendered invalid. The heuristics for auxiliary verbs and subject-verb agreement instead had to be modified to make use of the relevant dependency relations.

The source of this improvement can be seen even more clearly in Table 6.4, which breaks down the point-by-point agreement for the entire corpus into each individual score. Four more uninflected main verbs were identified correctly, while a whole *thirty* false positives were eliminated. Both false positives for adjunct infinitives (Sec3) were now correctly analyzed as complements (Sec5). Finally, although two new false negatives were introduced for the sentence point, thirty-one false positives were avoided.

Further significant improvements resulted from modifications to the rules determining whether verbs were properly conjugated. One such modification prevented incorrectly conjugated main verbs from being wrongly identified as gerunds or present participles, as in *The girl sitting there* in Lee's Chart 14; this is a common error in childhood language development, and thus needs to be scored accurately. Similarly, a rule was added to give attempt marks to negated or inverted verbs that lacked necessary auxiliary verbs. Another new rule used the included morphological analyzer (Boisclair, 2008) to determine whether irregular verbs were improperly conjugated, giving them an attempt mark if that was the case. Heuristics for subject-verb agreement were also improved to prevent the incorrect assignment of attempt marks for correctly conjugated verbs in certain cases.

Although none of the other additions or modifications to SYCORAX's rules produced as significant of an improvement on the Lee corpus, a number of them were noteworthy for solving problems relatively easily that would have required significantly more complex analyses without the addition of the parser. These include identifying the case of pronouns, which can be done simply by checking the dependency type in most cases; distinguishing the various uses of relative pronouns such as *that* and *who*, which can be done by identifying whether the verb to which they are attached is the main verb or a subordinate verb; and negatives, for which more than one cannot exist in the same clause.

For the most part, point-by-point agreement and correlation increased uniformly with each of these updates. One significant exception involved the analysis of conjunctions, as shown in Line 11 of Table 6.3; here, after optimizing the rules to use the parse tree, the

Table 6.4 **Accuracy of DSS before and after parser-based analysis of verbs.**

(a) Accuracy of DSS before addition of parsing.

Point	**Agr**	**Mis**	**Int**	**Tot**	**% Acc**
Indef1	37	0	1	38	97.3684
Indef3	17	0	5	22	77.2727
Indef4	2	0	0	2	100.0000
Indef7	2	1	0	3	66.6667
Pers1	47	0	0	47	100.0000
Pers2	65	0	8	73	89.0411
Pers3	39	0	1	40	97.5000
Pers6	1	0	2	3	33.3333
Pers7	0	0	2	2	0.0000
Main1	47	6	43	96	48.9583
Main2	51	3	6	60	85.0000
Main4	8	1	1	10	80.0000
Main6	8	0	0	8	100.0000
Main7	1	0	1	2	50.0000
Main8	1	0	0	1	100.0000
Sec2	5	0	0	5	100.0000
Sec3	0	0	2	2	0.0000
Sec4	0	0	2	2	0.0000
Sec5	3	2	0	5	60.0000
Sec7	1	0	0	1	100.0000
Sec8	1	0	0	1	100.0000
Neg4	6	0	1	7	85.7143
Neg5	2	0	0	2	100.0000
Neg7	8	0	1	9	88.8889
Conj3	2	0	0	2	100.0000
Conj5	1	0	0	1	100.0000
Conj8	5	0	1	6	83.3333
Rev1	1	2	0	3	33.3333
Rev4	1	0	0	1	100.0000
Rev6	5	0	0	5	100.0000
Wh2	7	3	0	10	70.0000
Wh5	1	0	0	1	100.0000
Wh7	3	0	0	3	100.0000
Sent1	90	1	70	161	55.9006
Total	468	19	147	634	73.8170

(b) Accuracy of DSS after parser-based analysis of verbs.

Point	**Agr**	**Mis**	**Int**	**Tot**	**% Acc**
Indef1	37	0	1	38	97.3684
Indef3	17	0	5	22	77.2727
Indef4	2	0	0	2	100.0000
Indef7	2	1	0	3	66.6667
Pers1	47	0	0	47	100.0000
Pers2	65	0	8	73	89.0411
Pers3	39	0	1	40	97.5000
Pers6	1	0	2	3	33.3333
Pers7	0	0	2	2	0.0000
Main1	51	2	13	66	77.2727
Main2	51	3	6	60	85.0000
Main4	8	1	1	10	80.0000
Main6	8	0	0	8	100.0000
Main7	1	0	1	2	50.0000
Main8	1	0	0	1	100.0000
Sec2	5	0	0	5	100.0000
Sec3					
Sec4	0	0	3	3	0.0000
Sec5	5	0	0	5	100.0000
Sec7	1	0	0	1	100.0000
Sec8	1	0	2	3	33.3333
Neg4	6	0	1	7	85.7143
Neg5	2	0	0	2	100.0000
Neg7	8	0	1	9	88.8889
Conj3	2	0	0	2	100.0000
Conj5	1	0	0	1	100.0000
Conj8	5	0	1	6	83.3333
Rev1	2	1	0	3	66.6667
Rev4	1	0	0	1	100.0000
Rev6	4	1	0	5	80.0000
Wh2	7	3	0	10	70.0000
Wh5	1	0	0	1	100.0000
Wh7	3	0	0	3	100.0000
Sent1	88	3	39	130	67.6923
Total	472	15	87	574	82.2300

correlation coefficient dropped by approximately 0.0002. This regression could be traced to a single sentence: *He said, "Where's my soup?"* The problem, in this case, is that *where* had originally been mistakenly identified as a conjunction, countering another error in the analysis of question inversion which was corrected later in the development process. Once *where* was correctly identified as an interrogative, the score decreased, thus causing the correlation to drop despite the improvement in point-by-point agreement.

6.2 Further Testing

Although it was a good starting point for tests, the Lee corpus did not provide a complete measure of SYCORAX's accuracy; specifically, these sample transcripts do not actually incorporate all of the constructions discussed in Lee's DSS rules. As a result, several improvements incorporated into SYCORAX based on Lee's description of the scoring of certain structures had no effect at all on the accuracy of SYCORAX on Lee's sample, as seen in lines 12–14 of Table 6.3. Among these omitted constructions were the discontinuous conjunction *whether or not*, adverbial uses of *more* and *most*, *that* as the subject of a subordinate clause, and elliptical deletions after interrogative adverbs (e.g., *I wonder why*).

Even beyond the rules specifically discussed in the DSS text, the Lee sample was a poor measure of accuracy for another reason. As the initial modifications to SYCORAX were tested using that sample, it essentially behaved as a training set; as with any machine learning algorithm, a separate data set from that used to optimize the application should ideally be used to evaluate it.

6.2.1 Scoring Challenges

The next obvious choice for a DSS testing corpus was that of Lively (1984). This paper discussed a number of common errors made by human DSS raters, giving examples of structures which have been incorrectly scored along with the correct scores for these structures. There was one slight problem, however, with evaluating the relative accuracy of SYCORAX and its competitors on this sample. For each example sentence, only the score for the particular structure under discussion is mentioned; neither the overall score for the sentence nor the full table of scores is given. Thus, because there is no agreed-upon reference score for any of these sentences, it is impossible to give an accurate correlation; for that matter, it is also inappropriate to use the standard formula for point-by-point agreement, as the automated score will naturally include a large number of intrusions in comparison.

For testing SYCORAX on the Lively sample, a modified version of point-by-point agreement was used, which will be termed *partial* point-by-point agreement. The key factor which allows this scoring method to be used is that the errors discussed by Lively fall into two groups: either a point that should be in the score is omitted or graded incorrectly, or a point that should *not* be present is incorrectly added to the score. Thus, in the former case, the point is said to agree if it is present in the automated score; in the latter case, the point is only said to agree if it is absent.

Although this is mostly identical to the normal measure of point-by-point agreement, there is one significant case in which it differs: the analysis of intrusions. In normal point-by-point agreement, if a point is omitted from both the reference score and the automated score, it is not counted in the number of agreements or the total number of points. In partial point-by-point agreement, on the other hand, omissions are made explicit in the reference score; thus, if a point is omitted from both the reference score and the automated score, it is counted as an agreement. Intrusions thus produce a decrease in accuracy in both measures of point-by-point agreement, but by different means; in normal point-by-point agreement, the denominator is increased, while in partial point-by-point agreement, the numerator is decreased.

6.2.2 Testing Agreement

A total of 96 sentences from the main text of Lively's paper were used for this test; this set comprised each list of sentences after the "Awarding the Sentence Point" heading, along with the sentence *We're supposed to go home now*, which was discussed but not specifically listed in the "Main Verbs" section. As several of these sentences specifically mention more than one point that may be analyzed incorrectly, these 96 sentences comprised a total of 115 points of agreement.

For this test, only SYCORAX and CP were evaluated; CLAN had already demonstrated itself to be significantly lower in its accuracy than CP on the relatively basic Lee sample, thus suggesting that it would perform with equally low if not lower accuracy on a more complex text. As the points under consideration applied to specific words and phrases, it was necessary to manually evaluate the partial point-by-point agreement to ensure that each matching point also applied to the correct word or phrase.

With a purely automated analysis, SYCORAX correctly identified 89 out of 115 points, or 77.3913%; CP identified only 68, or 59.1304%. Although the performance of SYCORAX was significantly better than that of CP, it was still nowhere near the 85% threshold for acceptability described by Long and Channell (2001).

Further investigation of the errors made by SYCORAX revealed that many of them were due to incorrect part-of-speech tagging; as JED was driven by part-of-speech tags, these tagging errors caused the sentences to also be parsed incorrectly. In particular, verbs were frequently mistaken for nouns in sentences such as *Marcia wants the one that rings*, *Did the boat turn over?*, and *Let the dog go*; however, no suitable rule could be found to improve the tagging of these sentences in ODT without an accompanying regression on noun phrases. To determine how much of the error was in fact the tagger's fault, SYCORAX was run on a manually retagged version of Lively's sample; the resulting accuracy was now 83.4783%, just less than two percent away from the threshold of acceptability.

6.2.3 Further Improving the Rules

Most of the errors that now remained in the scoring of the manually-tagged text were not due to inaccuracies in the parse tree, but instead resulted from omissions in the DSS rules. As a particularly embarrassing example, a rule to properly identify conjoined verbs with omitted auxiliaries, as discussed in Chapter 1, had yet to be implemented. Those changes which did require tweaks to the parser mostly involved special constructions such as *'d better* and adverbial *please*; however, two genuine bugs in the parser were also discovered, one involving infinitive dependents of gerunds and one involving a misinterpretation of question tags.

As expected, further improvements to the DSS rules, together with the aforementioned tweaks to the parser, significantly improved the performance on the Lively corpus, with accompanying improvements on the Lee corpus. These modifications are summarized in Table 6.5. Surprisingly, with suitable modifications to the parser and the DSS rules, it was possible to get the accuracy on the Lively corpus to exceed the 90% threshold even without modifications to the tagger.

Table 6.5 **Accuracy of automated SYCORAX DSS analysis during optimization for Lively corpus. Rows highlighted in gray required changes to the JED parser as well.**

		Lee All Unique	**Lee Correl**	**Lively**	**Lively Tagged**
1	Agree with object if subject is *here*	87.7395	0.978418	78.2609	84.3478
2	*me* requires plural verb	87.7395	0.978418	79.1304	85.2174
3	Attempt for *gotta* w/o *have*	87.7395	0.978418	81.7391	87.8261
4	*have* contractions	87.5240	0.979903	81.7391	87.8261
5	Ignore *'s* as *has* except with `VBN`	87.7395	0.978418	82.6087	88.6957
6	Rework question inflection rule	88.5277	0.979508	84.3478	90.4348
7	Identify auxiliaries on conjoined verbs	88.5277	0.979508	86.0870	92.1739
8	Identify passives using list from Lee p. 36	88.8889	0.981990	87.8261	93.9130
9	Mark conjoined passives	88.8889	0.981990	87.8261	93.9130
10	Add *put* to irregular verb lexicon	88.8889	0.981990	88.6957	94.7826
11	Parse the phrase *'d better*	88.8889	0.981990	89.5652	95.6522
12	Identify imperatives w/ adjuncts (e.g., *please*)	88.8889	0.981990	90.4348	96.5217
13	Prevent `VBG` head as subject of infinitive	88.8889	0.981990	91.3043	96.5217
14	Fix question tag mistaken for parenthetical	88.8889	0.981990	92.1739	97.3913

6.2.4 More Examples from Lively

In addition to the sentences discussed in the paper itself, Lively (1984) also included two appendices showing further examples, reproduced in Appendix D of this dissertation. The second of these appendices was a collection of 58 sentences, different from those given in the text itself, shown with the relevant subset of DSS scores. The first appendix, on the other hand, consisted of only ten sentences, but gave a full DSS table for those sentences. Clearly, it would be useful to test the performance of SYCORAX and CP on these two tables as well; in the case of Lively's Appendix A, it would even be possible to obtain a correlation coefficient.

The results of this experiment are shown in Table 6.6. Again, SYCORAX performed better than the two competing applications, even without further modification. Two minor tweaks to the parser, one allowing *to* following a preposition to be reinterpreted as an infinitive and one identifying *lemme* as a form of *let*, improved SYCORAX's partial agreement on Lively's Appendix B by an additional three percent.

Table 6.6 **Comparison of the accuracy of CP, CLAN, and SYCORAX on the appendices from Lively (1984).**

	CP	**CLAN**	**SYCORAX**	**SYCORAX + fixes**
Appendix A pt-by-pt	75.0000	21.3115	97.6190	97.6190
Appendix A correl	0.917578	0.839637	0.996553	0.996553
Appendix B pt-by-pt	68.7500	29.6875	87.5000	90.6250

6.3 A New Challenger: DSSA

When the above experiments were performed, Ron Channell's DSSA application (Channell, 2007), as reviewed in Judson (2006), had still not been released beyond Channell's own research group. However, on April 15, 2011, in response to an e-mail inquiry, Channell finally released a compiled binary of DSSA to the public, available at the URL given in the bibliography.

Along with DSSA, Channell also included a subset of the data which Judson used to evaluate the application. The included sample consists of two corpora: the Adam corpus, derived from interviews with a single child over a two-year span, and the Provo corpus, derived from a set of interviews with 30 children of ages two through seven. Both of these corpora included the manual DSS scores from a trained speech-language pathologist that were used in Judson's experiment. It was thus possible not only to evaluate SYCORAX against the state of the art in automated DSS, but also to compare the two applications' performance on a large, real-world corpus of speech transcripts with agreed-upon DSS scores.

6.3.1 Comparison on Lee and Lively

Before performing any experiments using the Judson data, however, it was worth testing DSSA on the samples used so far in this experiment. The results of this test are shown in Table 6.7; as can be seen, SYCORAX performs more accurately than DSSA on this collection of samples, even on the sentences from Lively.

Table 6.7 **Comparison of the accuracy of CP, SYCORAX, and DSSA on transcripts tested to this point.**

		CP	**SYCORAX**	**DSSA**
Point-by-Point Agreement	Lee 10	78.7097	98.4252	69.5035
	Lee 12	42.4528	71.4286	60.0000
	Lee 14	46.9027	77.0270	51.1111
	Lee 15	87.3016	94.2529	67.9558
	Lee 17	42.0000	88.4615	47.2222
	Lee 19	63.3663	89.0411	59.0909
	Lee All Unique	65.6652	88.8889	62.2673
	Lively "A"	75.0000	97.6190	70.8333
Partial Pt-by-Pt	Lively Main	68.7500	92.1739	66.0870
	Lively "B"	68.7500	90.6250	76.5625
Correlation Coefficient	Lee All Unique	0.931905	0.981990	0.922887
	Lively "A"	0.917578	0.996553	0.985326

Unlike both CP and CLAN, and like SYCORAX, DSSA performed as well when imperatives were not explicitly marked as when they were. Yet DSSA did have one significant disadvantage: like CLAN, but unlike CP and SYCORAX, DSSA did not attempt to guess whether sentences were grammatical in order to assign a sentence point. Instead, DSSA simply ignored the sentence point in its output entirely, effectively treating every sentence as ungrammatical and leaving this judgment up to human scorers instead. For a truly accurate comparison of both point-by-point agreement and correlation, therefore, each DSSA score for a valid sentence must have a sentence point added manually. Even with sentence points manually added, however, DSSA still performed worse than SYCORAX with no manual additions on all the samples tested so far, as can be seen in Table 6.8.

Table 6.8 **Comparison of the accuracy of unmodified SYCORAX scores and DSSA scores with sentence points added by hand.**

		SYCORAX	**DSSA + Sentence**
Point-by-Point Agreement	Lee 10	98.4252	85.8156
	Lee 12	71.4286	62.8571
	Lee 14	77.0270	62.2222
	Lee 15	94.2529	89.5028
	Lee 17	88.4615	61.1111
	Lee 19	89.0411	72.7273
	Lee All Unique	88.8889	76.8190
	Lively "A"	97.6190	83.3333
Partial Pt-by-Pt	Lively Main	92.1739	66.0870
	Lively "B"	90.6250	78.1250
Correlation Coefficient	Lee All Unique	0.981990	0.939713
	Lively "A"	0.996553	0.989444

6.3.2 The Adam and Provo Corpora

Clearly, SYCORAX outperforms DSSA on the set of tests from Lively designed to identify common errors in DSS scoring, as well as on the admittedly limited sample of real-world transcripts from Lee. However, for a true measure of accuracy, it is necessary to compare the two applications on a large and varied collection of transcripts—hence the Adam and Provo corpora that were bundled with DSSA.

The results from the Adam corpus were quite promising. As shown in Table 6.9, SYCORAX produced a more accurate correlation on the Adam corpus as a whole than DSSA with manually added sentence points, even when SYCORAX's automatically calculated sentence points were used; only five of the texts in the corpus correlated better with manual scores using DSSA than using SYCORAX, and one of those produced a better correlation using SYCORAX when manually-scored sentence points were also added to SYCORAX's scores. Using the default sentence points from SYCORAX, only three of the texts had higher point-by-point agreement than DSSA; however, when manually-scored sentence points were added to SYCORAX's scores, all but five transcripts exceeded DSSA's point-by-point agreement.

However, as seen in Table 6.10, SYCORAX performed significantly less accurately on the Provo corpus. Using SYCORAX's automated sentence points with no modification, only 11 of the 29 texts exceeded DSSA's correlation coefficient, and only one text exceeded DSSA's point-by-point agreement. The addition of manually-scored sentence points to SYCORAX's results caused only two more transcripts to exceed DSSA's correlation, and six more transcripts to exceed DSSA's point-by-point agreement.

6.3.3 Further Improvements to SYCORAX

Closer inspection of the results from the Adam and Provo corpora allowed a number of additional errors in SYCORAX's DSS rules to be identified and corrected. These modifications are summarized in Table 6.11, with corresponding overall accuracy scores on the Adam and Provo corpora.

Two key changes in particular led to significant improvements in SYCORAX's accuracy on the Adam and Provo corpora. The first such change involved contractions such as *'ll* and *'ve*, which were correctly attached in the parser but which were not properly identified in

Table 6.9 **Comparison between DSSA and SYCORAX scores on the Adam corpus. Gray highlights indicate an improvement over DSSA.**

	Point-By-Point			**Correlation**		
	DSSA + Sent	**SYCORAX**	**SYCORAX + Sent**	**DSSA + Sent**	**SYCORAX**	**SYCORAX + Sent**
Adam 36	85.5746	86.6180	92.2280	0.839821	0.880206	0.883257
Adam 37	85.5422	79.0244	84.0102	0.851570	0.857779	0.866751
Adam 38	83.2618	85.3659	87.8104	0.790086	0.916514	0.918923
Adam 39	83.7416	77.7056	83.9729	0.859786	0.881515	0.889592
Adam 40	87.0748	81.0934	87.6190	0.915782	0.950906	0.958789
Adam 41	95.0249	85.3774	90.5941	0.941596	0.959494	0.968697
Adam 42	87.5000	83.1486	88.6047	0.856892	0.889054	0.903062
Adam 43	88.0000	87.6214	90.8416	0.886837	0.888584	0.897218
Adam 44	85.9524	84.7500	88.3721	0.830894	0.903635	0.911394
Adam 45	80.3030	77.3987	83.6735	0.842335	0.819467	0.836657
Adam 46	90.3670	86.4560	89.9767	0.944875	0.934144	0.937405
Adam 47	83.6449	82.2844	88.0779	0.724864	0.844856	0.861120
Adam 48	83.5118	85.8093	88.0631	0.791822	0.916934	0.919274
Adam 49	86.0310	82.4834	87.2437	0.926788	0.898091	0.907742
Adam 50	83.9130	76.4957	81.4978	0.831360	0.895847	0.906609
Adam 51	80.0813	79.8755	85.1613	0.847619	0.915051	0.921590
Adam 52	88.0952	82.2650	87.2768	0.908945	0.798701	0.818179
Adam 53	82.4257	78.9474	85.7895	0.818064	0.797700	0.824515
Adam 54	82.6484	77.7027	83.0233	0.877317	0.907855	0.925334
Adam 55	88.7265	84.0580	89.0558	0.909425	0.942587	0.951685
Adam all	85.5287	82.1634	87.0862	0.863464	0.892719	0.903121

Table 6.10 **Comparison between DSSA and SYCORAX scores on the Provo corpus. Gray highlights indicate an improvement over DSSA.**

	Point-By-Point			Correlation		
	DSSA + Sent	SYCORAX	SYCORAX + Sent	DSSA + Sent	SYCORAX	SYCORAX + Sent
Aaron C	84.9910	84.2576	89.9621	0.874191	0.922273	0.934459
Aimee A	89.1514	78.5362	82.1338	0.957163	0.927185	0.932638
Alisha M	89.6936	82.4490	86.4146	0.923193	0.875200	0.884301
Amber B	92.1717	78.7515	84.2236	0.951528	0.810362	0.823936
Ambree J	89.8417	79.4451	84.3061	0.908062	0.870656	0.881808
Andrus R	79.0378	87.5686	91.4286	0.866746	0.896836	0.907311
Ashley A	86.8852	82.4000	87.8151	0.874532	0.845837	0.866856
Ashley B	91.9935	88.2927	91.3907	0.937610	0.907172	0.919424
BJ J	85.0080	82.6291	87.0340	0.832915	0.922373	0.930466
Christine B	90.4486	84.7242	90.0888	0.919794	0.852949	0.874345
Clarissa B	91.8990	76.6453	80.3645	0.964156	0.920003	0.923912
Cody B	90.9892	74.3979	78.9815	0.962766	0.886845	0.893926
Elizabeth	88.3754	76.8924	82.5243	0.940372	0.886040	0.889493
Heather B	89.7759	74.9676	80.3789	0.957585	0.914026	0.921586
Heather C	89.6403	85.7550	89.8975	0.873770	0.908040	0.918391
Jack M	85.0649	76.3975	81.8478	0.884217	0.857686	0.863506
Jarom	88.3149	77.8165	83.2927	0.918380	0.931510	0.938960
Katie K	87.1762	78.9855	84.7826	0.869315	0.904557	0.914533
Kevin B	86.6176	80.1712	85.0812	0.889506	0.860053	0.870969
Kyle K	79.2398	76.6871	83.9344	0.643142	0.861829	0.884671
Michael Z	87.6119	77.6389	83.9286	0.837653	0.890612	0.897455
Patrick A	84.7662	75.0720	80.7927	0.816938	0.858495	0.861526
Rebecca A	89.0724	78.9725	84.5960	0.922519	0.922398	0.929309
Rebecca T	90.3497	81.4111	85.5956	0.894710	0.907160	0.916440
Sarah	87.0108	75.7312	79.8538	0.956551	0.916642	0.918357
Scott	91.0000	83.9437	87.5714	0.880325	0.852363	0.867971
Talon	91.8981	90.4651	94.2446	0.848495	0.883867	0.904783
Tavida	86.5489	81.1828	84.1816	0.884511	0.882197	0.890726
Tiffany	87.6171	77.3216	81.2942	0.908634	0.881489	0.883859
Provo all	88.3054	79.6315	84.3263	0.924707	0.905879	0.912996

Table 6.11 **Iterations of SYCORAX development for the Judson corpora. Gray highlights indicate an improvement over DSSA; "default" and "manual" describe the source of the sentence points in SYCORAX's score.**

	Adam				Provo			
	Point-by-Point		Correlation		Point-by-Point		Correlation	
Revision	**Default**	**Manual**	**Default**	**Manual**	**Default**	**Manual**	**Default**	**Manual**
Initial version from Tables 6.9–6.10	82.1634	87.0862	0.892719	0.903121	79.6315	84.3263	0.905879	0.912996
Ignore fronted conjunctions	82.8940	87.8942	0.901392	0.912260	81.9616	86.8936	0.919672	0.927560
Identify contractions as auxiliaries	82.9549	87.9535	0.901935	0.912758	83.0811	87.5954	0.927244	0.932932
Count `VBP` mistagged as `VB`	83.2215	88.1372	0.904573	0.915162	83.3432	87.7846	0.927705	0.933281
Allow bare infinitives with *go*; introductory conjunctions before imperatives	83.3144	88.2200	0.904198	0.914833	83.4173	87.8258	0.927268	0.932713
Count infinitive with *go* as adjunct	83.3144	88.2200	0.904415	0.915033	83.4755	87.8879	0.927817	0.933290
Inversion applies to head verb, not first verb	83.4298	88.2784	0.906420	0.916010	83.5367	87.9150	0.928595	0.933750
Allow adverbs with imperatives, conjoined imperative	83.6927	88.4334	0.906743	0.916133	83.5910	87.9503	0.928607	0.933764
Agreement only applies to nearest subject dependent	83.8607	88.5674	0.908586	0.917633	83.8523	88.2259	0.929865	0.934938
Relative `WP` can agree universally with verbs	83.8607	88.5674	0.908586	0.917633	83.8568	88.2306	0.929869	0.934940
Reattach objects of subordinates as subjects of later verbs	83.8607	88.5674	0.908586	0.917633	83.8974	88.2758	0.930542	0.935607
Retag `NNS` after *one*, *this*, or *that* as `VBZ`	83.8607	88.5674	0.908586	0.917633	83.8936	88.2669	0.930644	0.935677
Reattach conjoined objects of prepositions as subjects of later verbs	83.8607	88.5674	0.908586	0.917633	84.1676	88.5302	0.932491	0.937277
Fix parsing of tag questions	84.1472	88.8022	0.908145	0.916877	84.1888	88.5291	0.931676	0.936419
Negation fixes; subject agreement for conjoined verbs	84.1154	88.7614	0.906589	0.915444	84.2289	88.5626	0.931849	0.936554
Count `VBD` mistaken for `VBN` as main verb	84.2111	88.8180	0.908792	0.917435	84.2825	88.5699	0.932433	0.936924
Ignore *how come* as a main verb	84.2494	88.8600	0.908606	0.917264	84.3071	88.6029	0.932152	0.936719
Improve tagging of `VBG`	84.1860	88.7773	0.904612	0.913452	84.3352	88.6340	0.933019	0.937637

DSS scoring; the correct scoring of these contractions allowed SYCORAX to produce a better overall correlation on the Provo corpus than DSSA even without any manual correction of sentence points.

The second change concerned a significant omission in the parser; specifically, the rule that reattached objects as subjects of later verbs failed to account for sentences where an apparent object of a preposition was actually the subject of a later verb (e.g., the word *it* in *I went to Athens and it was hot*). Correcting this rule allowed SYCORAX to exceed the point-by-point agreement of DSSA on both corpora when sentence points were manually corrected in both programs' output.

Several additional tweaks were made following this breakthrough in an attempt to make the DSS analysis even more accurate, as shown in the last five rows of Table 6.11, but none of them produced as significant an effect as these two prior changes. In the end, after this final set of modifications was complete, the correlation on the Adam corpus decreased slightly, though not enough to perform more poorly than DSSA; however, the point-by-point agreements for both corpora and the correlation for the Provo corpus increased, so these modifications were kept intact.

To complete the analysis, this improved version of SYCORAX was run again on the Lee and Lively samples. The results on these samples remained unchanged from those shown in Table 6.8, suggesting that the rules relevant to those sentences had remained stable.

6.4 Further Interpretation of Results

It was clear that overall, SYCORAX now had a slight advantage over DSSA in terms of accuracy on the corpora used by Judson (2006). Nonetheless, it was still not entirely clear how this improvement was distributed over the individual transcripts that make up these corpora, what specific syntactic structures contributed most to the improvement, or whether any structures produced a regression in accuracy compared to DSSA.

6.4.1 Per-Transcript Results

To identify how the improvement applied to the individual transcripts that make up the Judson samples, the experiment from Tables 6.9–6.10 was run again, this time using the revised version of SYCORAX; the per-transcript results from this run are presented in Tables 6.12 and 6.13.

Table 6.12 **Comparison between DSSA and SYCORAX scores on the Adam corpus, using a later revision of SYCORAX. Gray highlights indicate an improvement over DSSA.**

	Point-By-Point			Correlation		
	DSSA + Sent	SYCORAX	SYCORAX + Sent	DSSA + Sent	SYCORAX	SYCORAX + Sent
Adam 36	85.5746	86.4078	92.2280	0.839821	0.905888	0.906672
Adam 37	85.5422	79.9517	83.9196	0.851570	0.841537	0.851680
Adam 38	83.2618	86.1298	88.8128	0.790086	0.929473	0.931233
Adam 39	83.7416	80.4825	86.6972	0.859786	0.905668	0.915578
Adam 40	87.0748	85.0575	90.8434	0.915782	0.964108	0.973307
Adam 41	95.0249	88.2494	93.1990	0.941596	0.977099	0.982051
Adam 42	87.5000	84.4098	89.4860	0.856892	0.896201	0.910934
Adam 43	88.0000	88.5366	91.5212	0.886837	0.921505	0.924958
Adam 44	85.9524	85.1117	88.9460	0.830894	0.821333	0.831643
Adam 45	80.3030	77.2824	83.4842	0.842335	0.843013	0.860331
Adam 46	90.3670	88.9640	93.4118	0.944875	0.973173	0.975680
Adam 47	83.6449	84.2227	89.0777	0.724864	0.848843	0.865606
Adam 48	83.5118	86.5772	89.0661	0.791822	0.930009	0.931625
Adam 49	86.0310	82.8947	87.5566	0.926788	0.917783	0.925191
Adam 50	83.9130	80.7775	85.0780	0.831360	0.912097	0.920148
Adam 51	80.0813	81.5900	86.3341	0.847619	0.909528	0.913365
Adam 52	88.0952	84.2217	89.0380	0.908945	0.846220	0.861542
Adam 53	82.4257	81.2030	87.8307	0.818064	0.780231	0.808953
Adam 54	82.6484	84.1379	87.6485	0.877317	0.934371	0.943475
Adam 55	88.7265	88.3090	92.1909	0.909425	0.954220	0.959367
Adam all	85.5287	84.1860	88.7773	0.863464	0.904612	0.913452

Again, the fully automated point-by-point accuracy of SYCORAX still fell well below that of DSSA with manually added sentence points, with only seven of the Adam transcripts and four of the Provo transcripts exceeding the DSSA result. The fully automated correlation, however, showed significant improvement; though the same number of transcripts

Table 6.13 **Comparison between DSSA and SYCORAX scores on the Provo corpus, using a later revision of SYCORAX. Gray highlights indicate an improvement over DSSA.**

	Point-By-Point			Correlation		
	DSSA + Sent	**SYCORAX**	**SYCORAX + Sent**	**DSSA + Sent**	**SYCORAX**	**SYCORAX + Sent**
Aaron C	84.9910	88.6447	94.1748	0.874191	0.968379	0.976456
Aimee A	89.1514	85.5442	88.8401	0.957163	0.948249	0.951912
Alisha M	89.6936	85.9917	89.6996	0.923193	0.896327	0.905121
Amber B	92.1717	86.9619	91.2932	0.951528	0.904701	0.911589
Ambree J	89.8417	85.9375	89.6783	0.908062	0.916012	0.924096
Andrus R	79.0378	87.9121	92.1456	0.866746	0.913093	0.923636
Ashley A	86.8852	85.1626	90.3640	0.874532	0.914096	0.929915
Ashley B	91.9935	91.3681	93.5323	0.937610	0.945154	0.947780
BJ J	85.0080	87.0192	91.0000	0.832915	0.943179	0.949278
Christine B	90.4486	87.1105	91.9881	0.919794	0.896647	0.911079
Clarissa B	91.8990	83.6700	87.7622	0.964156	0.940198	0.944747
Cody B	90.9892	82.9222	87.2530	0.962766	0.916593	0.918480
Elizabeth	88.3754	80.7588	86.2020	0.940372	0.903855	0.902954
Heather B	89.7759	80.8824	85.4749	0.957585	0.935886	0.940588
Heather C	89.6403	88.7608	92.1481	0.873770	0.926030	0.933735
Jack M	85.0649	78.9144	84.3440	0.884217	0.887803	0.891817
Jarom	88.3149	82.4150	87.2840	0.918380	0.959058	0.963531
Katie K	87.1762	82.1867	87.8590	0.869315	0.936776	0.942308
Kevin B	86.6176	84.4380	88.3408	0.889506	0.904391	0.910254
Kyle K	79.2398	81.1912	87.5839	0.643142	0.911031	0.926046
Michael Z	87.6119	81.8830	88.3257	0.837653	0.914086	0.920162
Patrick A	84.7662	77.1596	83.0745	0.816938	0.895171	0.901080
Rebecca A	89.0724	82.0513	88.0674	0.922519	0.938695	0.944314
Rebecca T	90.3497	87.0396	89.7902	0.894710	0.928405	0.929972
Sarah	87.0108	81.9820	85.4008	0.956551	0.932060	0.933925
Scott	91.0000	88.7304	90.7381	0.880325	0.927026	0.930493
Talon	91.8981	91.3953	94.7242	0.848495	0.891481	0.908154
Tavida	86.5489	84.4049	87.6231	0.884511	0.915432	0.922400
Tiffany	87.6171	82.8283	86.6109	0.908634	0.916707	0.919573
Provo all	88.3054	84.3352	88.6340	0.924707	0.933019	0.937637

performed better on the Adam corpus, an additional nine transcripts from the Provo corpus now produced better correlation than DSSA.

With the sentence points in SYCORAX's score manually corrected, the improvement was even more impressive. Three more transcripts from the Adam corpus now produced better point-by-point agreement than DSSA, as did seven more transcripts from the Provo corpus. Although only one additional transcript from the Adam corpus correlated better with manual scores using SYCORAX than using DSSA, seven more of the Provo transcripts also produced better correlation using SYCORAX.

6.4.2 Per-Point Results

The other remaining question concerned whether SYCORAX was more or less accurate than DSSA on particular structures. Although the previous experiment shows that the improvement in SYCORAX's accuracy is reasonably generalized within each corpus, it does not provide any evidence as to which particular syntactic structures have contributed to this improvement.

However, one advantage of point-by-point agreement as a measure of accuracy is that it can be broken down into agreement scores for each individual point value, thus identifying the accuracy for each class of syntactic structure. This property was used in both Channell's evaluation of CP (2003) and Judson's evaluation of DSSA (2006).

A point-by-point breakdown of agreements, misses and intrusions is shown for the Adam corpus as a whole in Table 6.14, and for the Provo corpus in Table 6.15. In addition to the agreements, misses and intrusions for each application, the *change* in misses and intrusions between DSSA and SYCORAX is also shown; for these two scores, lower values are better, as this indicates improved accuracy in SYCORAX. In addition, each of these changes is also shown as a percentage of the total possible number of agreements. To better highlight where improvements have occurred, all negative values are presented with a gray background in the tables.

Table 6.14 **Comparison of agreements, misses and intrusions per point, between DSSA and SYCORAX on the Adam corpus.**

	DSSA			SYCORAX			Max	Δ	% Δ	Δ	% Δ
Point	**Agr**	**Mis**	**Int**	**Agr**	**Mis**	**Int**	**Agr**	**Miss**	**Miss**	**Int**	**Int**
Indef1	932	4	11	897	39	14	936	35	3.74	3	0.32
Indef3	351	1	24	342	10	7	352	9	2.56	-17	-4.83
Indef4	3	0	8	2	1	3	3	1	33.33	-5	-166.67
Indef7	27	0	4	24	3	7	27	3	11.11	3	11.11
Pers1	1449	6	16	1380	75	16	1455	69	4.74	0	0.00
Pers2	122	0	4	122	0	2	122	0	0.00	-2	-1.64
Pers3	187	0	1	180	7	1	187	7	3.74	0	0.00
Pers5	9	1	0	10	0	0	10	-1	-10.00	0	0.00
Pers6	47	5	6	33	19	13	52	14	26.92	7	13.46
Pers7	1	0	1	1	0	9	1	0	0.00	8	800.00
Main1	1091	33	312	999	125	49	1124	92	8.19	-263	-23.40
Main2	399	29	123	385	43	51	428	14	3.27	-72	-16.82
Main4	288	69	5	322	35	4	357	-34	-9.52	-1	-0.28
Main6	88	51	0	112	27	1	139	-24	-17.27	1	0.72
Main7	19	14	4	20	13	8	33	-1	-3.03	4	12.12
Main8	0	0	0	0	0	1	0	0	0.00	1	∞
Sec2	147	57	1	172	32	2	204	-25	-12.25	1	0.49
Sec3	1	9	0	6	4	14	10	-5	-50.00	14	140.00
Sec4	12	1	91	2	11	6	13	10	76.92	-85	-653.85
Sec5	99	24	41	97	26	8	123	2	1.63	-33	-26.83
Sec7	1	0	1	0	1	1	1	1	100.00	0	0.00
Sec8	10	10	0	10	10	26	20	0	0.00	26	130.00
Neg1	12	2	0	10	4	4	14	2	14.29	4	28.57
Neg4	123	17	1	136	4	7	140	-13	-9.29	6	4.29
Neg5	25	0	4	25	0	6	25	0	0.00	2	8.00
Neg7	60	0	24	56	4	12	60	4	6.67	-12	-20.00
Conj3	69	0	4	69	0	4	69	0	0.00	0	0.00
Conj5	42	1	3	38	5	3	43	4	9.30	0	0.00
Conj6	11	0	0	11	0	3	11	0	0.00	3	27.27
Conj8	74	50	12	102	22	21	124	-28	-22.58	9	7.26
Rev1	76	37	14	91	22	5	113	-15	-13.27	-9	-7.96
Rev4	4	11	0	14	1	6	15	-10	-66.67	6	40.00
Rev6	103	103	1	165	41	8	206	-62	-30.10	7	3.40
Rev8	0	1	0	0	1	0	1	0	0.00	0	0.00
Wh2	146	3	1	137	12	7	149	9	6.04	6	4.03
Wh5	38	9	8	45	2	9	47	-7	-14.89	1	2.13
Wh7	42	8	0	38	12	0	50	4	8.00	0	0.00
Wh8	6	0	0	5	1	0	6	1	16.67	0	0.00
Sent1	0	1457	0	1363	94	350	1457	-1363	-93.55	350	24.02
Non-Sent	6114	556	725	6058	612	338	6670	56	0.84	-387	-5.80
Total	6114	2013	725	7421	706	688	8127	-1307	-16.08	-37	-0.46

Table 6.15 **Comparison of agreements, misses and intrusions per point, between DSSA and SYCORAX on the Provo corpus.**

	DSSA			**SYCORAX**			**Max**	**Δ**	**% Δ**	**Δ**	**% Δ**
Point	**Agr**	**Mis**	**Int**	**Agr**	**Mis**	**Int**	**Agr**	**Miss**	**Miss**	**Int**	**Int**
Indef1	1938	11	26	1844	105	27	1949	94	4.82	1	0.05
Indef3	919	13	46	897	35	44	932	22	2.36	-2	-0.21
Indef4	6	0	1	6	0	1	6	0	0.00	0	0.00
Indef7	117	13	11	110	20	24	130	7	5.38	13	10.00
Pers1	2675	18	52	2628	65	30	2693	47	1.75	-22	-0.82
Pers2	855	16	9	826	45	4	871	29	3.33	-5	-0.57
Pers3	1295	3	11	1255	43	7	1298	40	3.08	-4	-0.31
Pers5	8	2	0	8	2	1	10	0	0.00	1	10.00
Pers6	87	18	21	67	38	29	105	20	19.05	8	7.62
Pers7	9	5	1	10	4	17	14	-1	-7.14	16	114.29
Main1	2403	73	544	2181	295	100	2476	222	8.97	-444	-17.93
Main2	1833	118	213	1754	197	126	1951	79	4.05	-87	-4.46
Main4	606	76	13	633	49	5	682	-27	-3.96	-8	-1.17
Main6	206	58	7	227	37	8	264	-21	-7.95	1	0.38
Main7	66	34	23	64	36	31	100	2	2.00	8	8.00
Main8	8	19	1	13	14	6	27	-5	-18.52	5	18.52
Sec2	305	136	4	360	81	55	441	-55	-12.47	51	11.56
Sec3	14	45	2	31	28	94	59	-17	-28.81	92	155.93
Sec4	11	5	135	4	12	31	16	7	43.75	-104	-650.00
Sec5	343	27	121	255	115	66	370	88	23.78	-55	-14.86
Sec7	5	8	0	2	11	1	13	3	23.08	1	7.69
Sec8	33	44	6	65	12	28	77	-32	-41.56	22	28.57
Neg1	34	11	0	33	12	3	45	1	2.22	3	6.67
Neg4	234	22	8	248	8	10	256	-14	-5.47	2	0.78
Neg5	27	3	1	28	2	2	30	-1	-3.33	1	3.33
Neg7	180	3	43	176	7	21	183	4	2.19	-22	-12.02
Conj3	547	5	5	544	8	5	552	3	0.54	0	0.00
Conj5	228	2	22	218	12	33	230	10	4.35	11	4.78
Conj6	80	0	1	80	0	10	80	0	0.00	9	11.25
Conj8	153	133	51	245	41	99	286	-92	-32.17	48	16.78
Rev1	105	28	20	105	28	8	133	0	0.00	-12	-9.02
Rev4	3	23	0	21	5	3	26	-18	-69.23	3	11.54
Rev6	92	94	5	146	40	8	186	-54	-29.03	3	1.61
Rev8	0	7	0	0	7	3	7	0	0.00	3	42.86
Wh2	175	22	7	178	19	4	197	-3	-1.52	-3	-1.52
Wh5	36	5	11	38	3	12	41	-2	-4.88	1	2.44
Wh7	18	11	0	14	15	0	29	4	13.79	0	0.00
Wh8	2	0	0	1	1	0	2	1	50.00	0	0.00
Sent1	0	3463	0	3302	161	889	3463	-3302	-95.35	889	25.67
Non-Sent	15656	1111	1421	15315	1452	956	16767	341	2.03	-465	-2.77
Total	15656	4574	1421	18617	1613	1845	20230	-2961	-14.64	424	2.10

These tables reveal a number of constructions for which SYCORAX notably outperforms DSSA, either by more frequently agreeing with manual scores or by reducing the detection of false positives:

- **Main verbs.** SYCORAX produces fewer false positives for simpler main verbs (`Main1–2`), and fewer false negatives for earlier compound verbs (`Main4–6`).
- **Early infinitives.** On both corpora, SYCORAX scores the use of early-developing infinitives such as *wanna* and *gotta* (`Sec2`) correctly more often.
- **Gerunds and participles.** Significantly fewer verbs are misdetected by SYCORAX as participles (`Sec4`) than by DSSA. In the Provo corpus, SYCORAX also identifies gerunds (`Sec8`) more often than DSSA; however, this does not hold for the Adam corpus.
- **Negations.** SYCORAX more often gives a correct score to *can't* and *don't* (`Neg4`), while preventing false positives more frequently for uncontracted negatives (`Neg7`).
- **Subordination.** SYCORAX correctly identifies 22% more subordinating conjunctions (`Conj8`) on the Adam corpus, and 32% more on the Provo corpus.
- **Question inversion.** For the simplest inversions (`Rev1`), SYCORAX produces fewer intrusions than DSSA, and on the Adam corpus, also correctly identifies 13% more of those that are present. Inversions of compound verbs with *be* (`Rev4`) are correctly identified two-thirds more often, and of verbs with modals and *do* (`Rev6`) are identified over a quarter more often.
- ***When* and *how*.** Both of these interrogative adverbs, scored as `Wh5`, are correctly identified more often than by SYCORAX than by DSSA, though the difference is more pronounced on the Adam corpus than on the Provo corpus.

Yet although there was a net improvement in accuracy, there are several structures, most notably pronouns and infinitives, for which SYCORAX performs significantly less accurately than DSSA. These inaccuracies, along with potential solutions, will be discussed in greater detail in the following chapter.

One additional point deserves attention, and that is the sentence point. The agreements shown in Tables 6.14–6.15 are directly derived from the two programs' output; unlike the prior experiments in this chapter, sentence points have not been corrected manually for either program. For only those sentences which should be given a sentence point, SYCORAX performs surprisingly accurately, identifying 93.55% within the Adam corpus and 95.35% within the Provo corpus. The problem in this case is with false positives: in both corpora, for approximately every four sentences correctly assigned a sentence point, one false positive was also produced by SYCORAX. As Tables 6.12–6.13 show, although these false positives have only a minor effect on the correlation of scores, they produce a nearly five percent decrease in overall point-by-point accuracy.

6.5 The Accuracy-Performance Tradeoff

As discussed previously, accuracy is only one aspect of the performance of automated DSS analyzers; the time and memory taken to perform an analysis cannot be ignored. The reason that a custom parser was written for use in SYCORAX was that state-of-the-art parsers were simply too inefficient compared to existing automated DSS utilities to make up for the expected increase in accuracy.

Obtaining truly accurate timings for these programs' DSS analyses proved to be impossible. Not only does each of these programs run under a different runtime environment, but the process from the perspective of the user is significantly different in each of these applications:

- In SYCORAX, a Windows .NET application, the text is tagged and then parsed; then, the resulting parse tree is analyzed by a collection of DSS rules. This entire process is activated by a single click, so that a DSS table is directly produced from an input sentence; if necessary, options are specified via checkboxes before running the analysis.

- DSSA, a Mac OS X application, is most similar in design to SYCORAX. After the user selects a file, processing begins, and the DSS score is then output with no further user intervention.
- In CLAN, a Windows C++ application, the text must be morphologically analyzed, disambiguated, and then analyzed for DSS. Each of these steps requires a separate command to be entered into CLAN's command line, but it is possible to run several commands as a batch.
- In CP, a MS-DOS application, the text must first be converted into a CORPUS file. A LARSP analysis is then generated from the CORPUS file, and then a DSS analysis is produced from the LARSP file. Each of these steps requires running a separate sub-program from CP's menu and entering appropriate parameters at prompts.

Given the vastly different environments and the vastly different processes necessary to produce a DSS analysis, it was only reasonable that wall-clock time should instead be used as a measure of performance—specifically, the amount of time necessary to perform the entire process generating a DSS score from an input transcript.

6.5.1 Initial Testing

For the following set of tests, the same input files were used: all 201 sentences, including repetitions, contained within the six charts from Lee (1974), along with the 96 sentences from Lively (1984). All tests were run on a MacBook with a 2.4 GHz Core 2 Duo processor. For the three Windows applications, tests were run in a Windows XP Mode virtual machine under Windows 7 64-bit; running the programs natively in Windows 7 would have been the best option, of course, but this was impossible given the aforementioned incompatibility of CP. As it was a Macintosh application, DSSA was simply run using the stock installation of Mac OS X 10.6.

During an initial test of SYCORAX, an odd discrepancy in timing became apparent; the first analysis in any SYCORAX execution took approximately 0.7 seconds longer than any analysis afterward. The reason for this delay was that the parser and analyzer were only compiled into native code and loaded into memory by the .NET framework's just-in-time compiler at the start of the first analysis. To improve the program's apparent performance from the end user's perspective, the tagger and parser were primed immediately after the tagger's lexicon was loaded.

With this modification made, the execution times for SYCORAX are shown in Table 6.16. Clearly, the greatest bottleneck in the case of SYCORAX is the initialization phase; once initialization is complete, a transcript of a hundred sentences can be analyzed in a fraction of a second.

Table 6.16 **Initialization and analysis times for SYCORAX, after adding priming of analyzer.**

	Seconds to run					
	1	**2**	**3**	**4**	**5**	**Avg**
Initialize	6.35	6.28	6.33	6.21	6.04	6.24
Analyze Lee	0.89	0.57	0.67	0.70	0.64	0.69
Analyze Lee again	0.55	0.55	0.68	0.66	0.52	0.59
Analyze Lively	0.56	0.58	0.59	0.58	0.59	0.58
Analyze Lively again	0.53	0.47	0.57	0.52	0.52	0.52

For the purpose of comparison, Computerized Profiling, CLAN and DSSA were all run five times on the same collection of sentences from Lee. The wall-clock time taken for each analysis is shown in Table 6.17.

As described previously, obtaining a truly accurate comparison of the time taken by just the DSS algorithms is impossible, especially for Computerized Profiling; as discussed in Chapter 5, that program's interface is designed so as not to allow the steps of the analysis to be performed as a batch. With this caveat in mind, the time taken by SYCORAX still appears to be extremely competitive, especially in comparison to DSSA, the most accurate

Table 6.17 **Analysis times for Computerized Profiling, CLAN, and DSSA, from raw transcript to DSS output.**

	Seconds to run					
	1	**2**	**3**	**4**	**5**	**Avg**
Computerized Profiling	19.46	20.23	20.62	18.96	19.35	19.72
CLAN	5.46	4.96	3.45	3.59	3.45	4.18
DSSA	0.59	0.58	0.54	0.72	0.60	0.61

of its competitors by far. This is particularly significant given that SYCORAX was run in a virtualized environment while DSSA was run natively.

6.5.2 Further Tests

To determine whether this pattern would still hold for an even larger input, both SYCORAX and DSSA were run on a combination of the first ten transcripts from the Adam corpus, which comprise a total of 1,036 sentences. In this case, because CLAN and CP were not being evaluated, both SYCORAX and DSSA were run natively rather than virtualized.

The results for this experiment are shown in Table 6.18. Although it initially appeared that SYCORAX was significantly slower than DSSA, by a factor of about 15, this turned out to be entirely a side effect of the Windows Forms GUI. Buffering the output to the GUI caused the run time to be slightly faster than DSSA on average, as also shown in Table 6.18.

Table 6.18 **Analysis times for DSSA and SYCORAX on a 1,036-sentence subset of the Adam corpus.**

	Seconds to run					
	1	**2**	**3**	**4**	**5**	**Avg**
DSSA	1.91	1.93	1.99	1.99	1.91	1.95
SYCORAX	31.63	30.67	30.60	...	...	30.97
SYCORAX w/ Buffer	2.09	1.58	1.61	1.54	1.59	1.68

One final concern remained regarding performance, and that was memory usage. To measure this, both DSSA and SYCORAX were run five times on the same subset of the Adam corpus without exiting either program, and the memory usage was recorded at the peak of each analysis for both programs. The results of this experiment are shown in Table 6.19.

Table 6.19 **Memory usage, in megabytes, for DSSA and SYCORAX on a 1,036-sentence subset of the Adam corpus.**

	Megabytes of RAM					
	1	**2**	**3**	**4**	**5**	**Avg**
DSSA	18.9	18.8	19.7	20.6	20.5	19.7
SYCORAX	138	187	241	...	...	189
SYCORAX w/ Buffer	112	130	145	165	183	147

Unfortunately, memory use was the one aspect in which SYCORAX did not outperform its competitors; it consistently used almost ten times as much RAM as DSSA. This appeared to be the result of a memory leak somewhere in the program; while DSSA's memory usage went up and down, SYCORAX's memory consumption continued to increase after each analysis. Nonetheless, even with this memory leak, SYCORAX's use of RAM was still significantly smaller than it would have been if a full-fledged parser such as MaltParser had been incorporated.

Clearly, then, JED is a worthwhile compromise for use in SYCORAX: it allows the DSS analyzer to identify certain structures which a purely linear analysis could not, with memory consumption significantly lower than that of an exhaustive parser, and with execution times competitive with those of linear DSS analyzers.

Chapter 7

Known Issues and Future Work

Although SYCORAX did perform more poorly on a number of transcripts than DSSA, its overall performance on the Adam and Provo corpora, together with its performance on the Lee and Lively data sets, provide strong evidence that the addition of a shallow parser, combined with DSS rules that are able to make use of the parse tree, can improve the accuracy of automated DSS with no significant effect on execution time. Yet there are still aspects of SYCORAX which could benefit from further improvement.

First, there are still problems with SYCORAX's accuracy. Of course, there is the aforementioned issue of sentence points; however, even with the grammaticality of sentences scored manually, several other constructions frequently produce false negatives or false positives when automatically scored by SYCORAX.

Second, there are questions regarding the accuracy of the manual scores which have been assigned to the corpora used by Judson (2006). Although grading was performed by speech-language pathologists trained in DSS, and although inter-rater reliability between two raters was found to be between 95% and 97% by Judson (2006), I have found a number of sentences which are clearly scored incorrectly based solely on examples given in Lee's guidelines.

Finally, though it does not affect the accuracy of SYCORAX, there is the problem of memory consumption, as discussed in the previous chapter. Although SYCORAX is on par with its competition with respect to timing, and slightly better with respect to accuracy, this improved accuracy comes at the expense of a greater RAM footprint. There is a clear need to optimize SYCORAX's memory use so as to make it more efficient in that respect.

7.1 Errors Made by SYCORAX

Although SYCORAX did produce DSS scores with greater accuracy than DSSA on a variety of tests, as shown in Table 6.14–6.15 (pages 122–123), a number of syntactic structures are still scored more poorly by SYCORAX than by DSSA and thus deserve further attention.

Pronouns, in particular, are a frequent source of error: although there were fewer intrusions for most indefinite and personal pronoun scores, there are an even greater number of pronouns in the Adam and Provo corpora which were correctly identified by DSSA but not by SYCORAX, for two main reasons. A majority of the misses for indefinite pronouns occur on the `Indef1` score; most of these can be traced to incorrect tagging of the word *that*, which, as discussed previously, can be given three different scores based on context. Personal pronouns, on the other hand, were often missed by SYCORAX because of mistaken judgments about subject-verb agreement, usually resulting from incorrect parsing; here, additional improvements to JED's rules could potentially be beneficial.

Another common problem which deserves further attention is the distinction between complementing and adjunct infinitives. In both corpora, SYCORAX frequently mistook complementing infinitives (`Sec5`) for adjuncts (`Sec3`), producing intrusions on the latter score and, in the case of the Provo corpus, numerous misses on the former. This, too, could be solved through additional tweaks to the two infinitive attachment rules in the parser.

Finally, although this was not one of the structures for which SYCORAX had a net decrease in accuracy, there is a prevalent error in the `Main1` score that involves the contraction *'s*, which is often incorrectly tagged as a possessive by ODT. This is one of the main problems with a tag-driven parser such as JED: if the tags are in any way incorrect, the resulting parse will also be incorrect. To correct all of the misses involving *'s*, it would be necessary either to improve ODT's rules to better disambiguate *'s*, or else to perform the disambiguation in the parser itself.

7.2 Errors in Manual Scores

Furthermore, some of the errors in the Provo corpus may not be due to the automated DSS analysis at all; rather, they may be purely the fault of the *human* analyzers whose work Judson used.

Ten errors were discovered in the manual scores for the Provo corpus during a cursory glance at the results from SYCORAX; these are presented below in Table 7.1. Admittedly, this is a small sample which makes up only 0.2% of the entire corpus; nonetheless, to ensure that accuracy is measured reliably, it would be worthwhile to check the manual scores for any other significant inaccuracies and correct any errors that were found.

Table 7.1 **A selection of ten errors in the manual scores of the Provo corpus.**

Sentence	**Scored As**	**Should Be**
I did it.	Main1	Main2
My mom help put these one on.	Main1	Attempt
My mom help put these on.	Main1	Attempt
My mom help put them on.	Main1	Attempt
It's not the tape I want.	Pers1	Indef1
It's a eyes.	Pers1	Indef1
There's guns here.	Main1	Attempt
Where's their magnets?	Main1	Attempt
I gotta look for it.	Main2	Attempt
You didn't like it?	Rev6	None

7.3 Memory Consumption

Finally, it is necessary to address the memory consumption of SYCORAX. While the 200-megabyte RAM footprint of SYCORAX is significantly better than that of full-fledged state-of-the-art parsers, it nonetheless seems quite extreme in comparison to the memory consumption of SYCORAX's closest competitors.

Curiously, the greatest memory leak in SYCORAX comes from neither the parser nor the DSS analysis algorithm, but rather, from the Windows Forms GUI. Again, this can be conclusively demonstrated through an experiment using a command-line version of SYCORAX, which was trivial to develop due to the fact that SYCORAX's model lies in a separate library from the view and controller. Running this experimental command-line version of SYCORAX on the same input used in Table 6.19 (page 129) stabilized at 80 megabytes of RAM after approximately ten seconds of execution.

Furthermore, the bulk of *this* memory usage lies in the tagger. Simply initializing the ODT library, with no further analysis performed, uses 55 megabytes of RAM in itself. This is no doubt due to the fact that ODT's lexicon is a 16-megabyte text file when decompressed—and is stored in RAM by ODT in an even less compact format, using a 32-bit integer for each tag frequency when most frequencies are less than a single digit in size.

Clearly, SYCORAX could use some further optimization to make it even more memory-efficient; optimizing the GUI code and using a more efficient data structure to store the lexicon would both produce significant drops in RAM consumption.

Bibliography

Abney, Steven P. 1991. Parsing by chunks. In *Principle-based parsing*, ed. Robert Berwick, Steven Abney, and Carol Tenny, 257–278. New York: Kluwer Academic Publishers.

Boisclair, Cody. 2008. Developing a tokenizer and morphological parser for English text in C#. In *Proceedings of the 46th Annual Southeast Regional Conference*, 288–293. ACM.

Botel, Morton, and Alvin Granowsky. 1972. A formula for measuring syntactic complexity: A directional effort. *Elementary English* 49(4):513–516.

Brill, Eric. 1994. *Rule based tagger* [Computer program]. Cambridge, MA: Massachusetts Institute of Technology. `http://www.tech.plym.ac.uk/soc/staff/guidbugm/software/RULE_BASED_TAGGER_V.1.14.tar.Z` (accessed 25 January 2010).

———. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics* 21(4):543–565.

Brown, Cati, Tony Snodgrass, Susan J. Kemper, Ruth Herman, and Michael A. Covington. 2008. Automatic measurement of propositional idea density from part-of-speech tagging. *Behavior Research Methods* 40(2):540–545.

Buchholz, Sabine, and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, 149–164. Association for Computational Linguistics.

Channell, Ron W. 2003. Automated Developmental Sentence Scoring using Computerized Profiling software. *American Journal of Speech-Language Pathology* 12(3):369–375.

———. 2007. *DSSA* [Computer program], version 0.99. Provo, UT: Brigham Young University. http://comd.byu.edu/rc/gcs.htm (accessed 27 April 2011).

Cheung, Hintat, and Susan Kemper. 1992. Competing complexity metrics and adults' production of complex sentences. *Applied Psycholinguistics* 13(1):53–76.

Chomsky, Carol. 1986. Analytic study of the Tadoma method: Language abilities of three deaf-blind subjects. *Journal of Speech and Hearing Research* 29(3):332.

Covington, M. A., W. J. Riedel, C. Brown, C. He, E. Morris, S. Weinstein, J. Semple, and J. Brown. 2009. Ketamine and schizophrenic speech: More difference than originally reported. *Journal of Psychopharmacology* 23(1):111–112.

Covington, Michael A. 1990. A dependency parser for variable-word-order languages. Tech. Rep., University of Georgia. http://www.ai.uga.edu/ftplib/ai-reports/ai199001.pdf (accessed 2 August 2010).

———. 1994. *Natural language processing for Prolog programmers*. Englewood Cliffs, NJ: Prentice Hall.

———. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, 95–102.

Covington, Michael A., Congzhou He, Cati Brown, Lorina Naçi, and John Brown. 2006. How complex is that sentence? A proposed revision of the Rosenberg and Abbeduto D-Level scale. Tech. Rep., University of Georgia. http://www.ai.uga.edu/caspr/2006-01-Covington.pdf (accessed 4 May 2009).

Crystal, David, Paul Fletcher, and Michael Garman. 1989. *Grammatical analysis of language disability*. San Diego: Singular Pub. Group.

DeThorne, Laura S., Stephen A. Petrill, Sara A. Hart, Ron W. Channell, Rebecca J. Campbell, Kirby Deater-Deckard, Lee Anne Thompson, and David J. Vandenbergh. 2008. Genetic effects on children's conversational language use. *Journal of Speech, Language, and Hearing Research* 51(2):423.

Engelman, Michal, Emily M. Agree, Lucy A. Meoni, and Michael J. Klag. 2010. Propositional density and cognitive function in later life: Findings from the Precursors Study. *The Journals of Gerontology Series B: Psychological Sciences and Social Sciences* 65B(6):706–711.

Finestack, Lizbeth H., and Leonard Abbeduto. 2010. Expressive language profiles of verbally expressive adolescents and young adults with Down syndrome or Fragile X syndrome. *Journal of Speech, Language, and Hearing Research* 53(5):1334–1348.

Frazier, Lyn. 1985. Syntactic complexity. In *Natural language parsing: psychological, computational, and theoretical perspectives*, ed. David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, chap. 4. Cambridge: Cambridge University Press.

Hewitt, Lynne E., Carol Scheffner Hammer, Kristine M. Yont, and J. Bruce Tomblin. 2005. Language sampling for kindergarten children with and without SLI: Mean length of utterance, IPSYN, and NDW. *Journal of Communication Disorders* 38(3):197–213.

Holdgrafer, Gary. 1995. Comparison of two methods for scoring syntactic complexity. *Perceptual and Motor Skills* 81(2):498.

Huddleston, Rodney D., Geoffrey K. Pullum, and Laurie Bauer. 2002. *The Cambridge grammar of the English language*. Cambridge: Cambridge University Press.

Hudson, Richard. 1989. Towards a computer-testable Word Grammar of English. *UCL Working Papers in Linguistics* 1:321–339.

Hughes, Diana L., Marc E. Fey, and Steven H. Long. 1992. Developmental Sentence Scoring: Still useful after all these years. *Topics in Language Disorders* 12(2):1–12.

Jarrold, William L., Bart Peintner, Eric Yeh, Ruth Krasnow, Harold S. Javitz, and Gary E. Swan. 2010. Language analytics for assessing brain health: Cognitive impairment, depression and pre-symptomatic Alzheimer's disease. In *Brain Informatics*, 299–307. Berlin: Springer.

Järvinen, Timo, and Pasi Tapanainen. 1997. A dependency parser for English. Tech. Rep. TR-1, University of Helsinki. `http://web.archive.org/web/20061209031144/www.ling.helsinki.fi/~tapanain/dg/doc/index.html` (accessed 2 August 2010).

Johnson, Martha R., and J. Bruce Tomblin. 1975. The reliability of Developmental Sentence Scoring as a function of sample size. *Journal of Speech and Hearing Research* 18(2):372.

Judson, Carrie. 2006. Accuracy of automated Developmental Sentence Scoring software. Master's thesis, Brigham Young University. `http://contentdm.lib.byu.edu/ETD/image/etd1448.pdf` (accessed 25 October 2009).

Kemper, Susan, Ruth Herman, and Cindy Lian. 2003. Age differences in sentence production. *Journals of Gerontology Series B: Psychological Sciences and Social Sciences* 58(5):260–268.

Kemper, Susan, Ruth E. Herman, and Chiung-Ju Liu. 2004. Sentence production by young and older adults in controlled contexts. *Journals of Gerontology Series B: Psychological Sciences and Social Sciences* 59(5):P220–224.

Kemper, Susan, Emily LaBarge, F. Richard Ferraro, Hintat Cheung, Him Cheung, and Martha Storandt. 1993. On the preservation of syntax in Alzheimer's disease: Evidence from written sentences. *Archives of Neurology* 50(1):81–86.

Kernan, Keith T., and Sharon Sabsay. 1996. Linguistic and cognitive ability of adults with Down syndrome and mental retardation of unknown etiology. *Journal of Communication Disorders* 29(5):401–422.

Kintsch, Walter, and Janice Keenan. 1973. Reading rate and retention as a function of the number of propositions in the base structure of sentences. *Cognitive Psychology* 5(3):257–274.

Klee, Thomas, and Elizabeth Sahlie. 1986. Review of *DSS Computer Program*, by Peter K. Hixson. *Child Language Teaching and Therapy* 2(2):231–235.

Klein, Dan, and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, 423–430. Association for Computational Linguistics.

Lee, Laura Louise. 1974. *Developmental sentence analysis: A grammatical assessment procedure for speech and language clinicians*. Evanston: Northwestern University Press.

Lively, Mary Ann. 1984. Developmental Sentence Scoring: Common scoring errors. *Language, Speech, and Hearing Services in Schools* 15(3):154.

Long, Steven H., and Ron W. Channell. 2001. Accuracy of four language analysis procedures performed automatically. *American Journal of Speech-Language Pathology* 10(2):180–188.

Long, Steven H., Marc E. Fey, and Ron W. Channell. 2006. *Computerized profiling* [Computer program], version 9.7.0. Milwaukee: Department of Speech Pathology and Audiology, Marquette University. `http://computerizedprofiling.org/`.

Lyons, Kelly, Susan Kemper, Emily Labarge, F. Richard Ferraro, David Balota, and Martha Storandt. 1994. Oral language and Alzheimer's disease: A reduction in syntactic complexity. *Aging, Neuropsychology, and Cognition* 1(4):271–281.

MacWhinney, Brian. 2000. The CLAN programs. In *The CHILDES project: Tools for analyzing talk*, vol. 1, 3rd ed., chap. 2. Mahwah, NJ: Lawrence Erlbaum Associates.

———. 2011. *CLAN* [Computer program], version 11-Mar-2011. Pittsburgh, PA: Carnegie Mellon University. `http://childes.psy.cmu.edu/clan/`.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2):330.

Marcus, Mitchell P., Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. 1999. *Treebank-3* [CD-ROM]. Philadelphia: Linguistic Data Consortium. LDC Catalog No. LDC99T42.

Mayberry, Rachel I. 1973. Oral and manual language skills of hearing children of deaf parents. *Independent Studies and Capstones.* Paper 147. Program in Audiology and Communication Sciences, Washington University School of Medicine. `http://digitalcommons.wustl.edu/pacs_capstones/147/` (accessed 4 March 2011).

McDonald, Ryan, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency parsing with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, 216–220.

Minch, Stacy Lynn. 2009. Validity of seven syntactic analyses performed by the Computerized Profiling software. Master's thesis, Brigham Young University. `http://contentdm.lib.byu.edu/ETD/image/etd2956.pdf` (accessed 17 May 2011).

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, 932. Association for Computational Linguistics.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007b. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering* 13(2):95–135.

Pierce, Sandra, and Giampiero Bartolucci. 1977. A syntactic investigation of verbal autistic, mentally retarded, and normal children. *Journal of Autism and Developmental Disorders* 7(2):121–134.

Politzer, Robert L. 1974. Developmental Sentence Scoring as a method of measuring second language acquisition. *Modern Language Journal* 58(5):245–250.

Pye, Clifton. 1994. Review of *The CHILDES Project: Tools for Analyzing Talk*, by Brian MacWhinney. *Language* 70(1):156–159.

Rescorla, Leslie, Katherine Dahlsgaard, and Julie Roberts. 2000. Late-talking toddlers: MLU and IPSyn outcomes at 3;0 and 4;0. *Journal of Child Language* 27(03):643–664.

Rosenberg, Sheldon, and Leonard Abbeduto. 1987. Indicators of linguistic competence in the peer group conversational behavior of mildly retarded adults. *Applied Psycholinguistics* 8(1):19–32.

Santorini, Beatrice. 1995. Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision, 2nd printing). Tech. Rep. MS-CIS-90-47, University of Pennsylvania. `ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz` (accessed 11 January 2010).

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

Snowdon, D. A., L. H. Greiner, and W. R. Markesbery. 2000. Linguistic ability in early life and the neuropathology of Alzheimer's disease and cerebrovascular disease: Findings from the Nun Study. *Annals of the New York Academy of Sciences* 903(1):34–38.

Snowdon, David A., Susan J. Kemper, James A. Mortimer, Lydia H. Greiner, David R. Wekstein, and William R. Markesbery. 1996. Linguistic ability in early life and cognitive function and Alzheimer's disease in late life: Findings from the Nun Study. *JAMA* 275(7):528–532.

Søgaard, Anders, and Jonas Kuhn. 2009. Using a maximum entropy-based tagger to improve a very fast vine parser. In *Proceedings of the 11th International Conference on Parsing Technologies*, 206–209. Association for Computational Linguistics.

Tsai, Shu-Chiao. 2010. Developing and integrating courseware for oral presentations into ESP learning contexts. *Computers & Education* 55(3):1245–1258.

Voss, Matthew J. 2005. Determining syntactic complexity using very shallow parsing. Master's thesis, University of Georgia. `http://www.ai.uga.edu/iai/theses/voss_matthew.pdf` (accessed 18 August 2009).

Yngve, Victor H. 1960. A model and an hypothesis for language structure. *Proceedings of the American Philosophical Society* 104(5):444–466.

Appendix A

Penn Treebank Tag Set

The following table, defining all part-of-speech tag symbols, is reproduced from the Penn Treebank tagging guidelines (Santorini, 1995).

Tag	**Part of speech**
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle

(Continued on next page)

(Continued from previous page)

Tag	**Part of speech**
`VBN`	Verb, past participle
`VBP`	Verb, non-3rd person singular present
`VBZ`	Verb, 3rd person singular present
`WDT`	Wh-determiner
`WP`	Wh-pronoun
`WP$`	Possessive wh-pronoun
`WRB`	Wh-adverb

Appendix B

Summary of DSS Scoring Rules

The following two-page table, reproduced from Lee (1974), summarizes the entire set of DSS scoring rules as proposed by Lee; the numbers on the left represent the point values given to constructs, while the labels on top identify the columns in which constructs are scored. Thus, for instance, the words *it*, *this* and *that* each score one point in the Indefinite Pronouns column, while *because* is given six points in the Conjunctions column.

	Indefinite Pronouns or Noun Modifiers	Personal Pronouns	Main Verbs	Secondary Verbs
1	it, this, that	1st and 2nd person: I, me, my, mine, you, your(s)	A. Uninflected verb: I *see* you. B. copula, is or 's: *It's* red. C. is + verb + ing: He *is coming.*	
2		3rd person: he, him, his, she, her, hers	A. -s and -ed: *plays, played* B. irregular past: *ate, saw* C. Copula: *am, are, was, were* D. Auxiliary *am, are, was, were*	Five early-developing infinitives: I wan*na see* (want *to see*) I'm gon*na see* (going *to see*) I got*ta see* (got *to see*) Lemme [to] see (let me [*to*] *see*) Let's [to] play (let [us *to*] *play*)
3	A. no, some, more, all, lot(s), one(s), two (etc.), other(s), another B. something, somebody, someone	A. Plurals: we, us, our(s), they, them, their B. these, those		Non-complementing infinitives: I stopped *to play*. I'm afraid *to look*. It's hard *to do* that.
4	nothing, nobody, none, no one		A. can, will, may + verb: *may go* B. Obligatory do + verb: *don't go* C. Emphatic do + verb: *I do see.*	Participle, present or past: I see a boy *running*. I found the toy *broken*.
5		Reflexives: myself, yourself, himself, herself, itself, themselves		A. Early infinitival complements with differing subjects in kernels: I want you *to come*. Let him *[to] see*. B. Later infinitival complements: I had *to go*. I told him *to go*. I tried *to go*. He ought *to go*. C. Obligatory deletions: Make it *[to] go*. I'd better *[to] go*. D. Infinitive with wh-word: I know what *to get*. I know how *to do* it.
6		A. Wh-pronouns: who, which, whose, whom, what, that, how many, how much I know *who* came. That's *what* I said. B. Wh-word + infinitive: I know *what* to do. I know *who(m)* to take.	A. could, would, should, might + verb: *might come, could be* B. Obligatory does, did + verb C. Emphatic does, did + verb	
7	A. any, anything, anybody, anyone B. every, everything, everybody, everyone C. both, few, many, each, several, most, least, much, next, first, last, second (etc.)	(his) own, one, oneself, whichever, whoever, whatever Take *whatever* you like.	A. Passive with *get*, any tense Passive with *be*, any tense B. must, shall + verb: *must come* C. have + verb + en: *I've eaten* D. have got: I*'ve got* it.	Passive infinitival complement: With *get*: I have *to get dressed.* I don't want *to get hurt*. With *be*: I want *to be pulled*. It's going *to be locked*.
8			A. have been + verb + ing had been + verb + ing B. modal + have + verb + en: *may have eaten* C. modal + be + verb + ing: *could be playing* D. Other auxiliary combinations: *should have been sleeping*	Gerund: *Swinging* is fun. I like *fishing*. He started *laughing*.

	Negatives	Conjunctions	Interrogative Reversals	Wh-Questions
1	it, this, that + copula or auxiliary is, ’s + not: It’s *not* mine. This is *not* a dog. That is *not* moving.		Reversal of copula: *Isn’t it* red? *Were they* there?	
2				A. who, what, what + noun: *Who* am I? *What* is he eating? *What book* are you reading? B. where, how many, how much, what...do, what...for *Where* did it go? *How much* do you want? *What* is he *doing*? *What* is a hammer *for*?
3		and		
4	can’t, don’t		Reversal of auxiliary be: *Is he* coming? *Isn’t he* coming? *Was he* going? *Wasn’t he* going?	
5	isn’t, won’t	A. but B. so, and so, so that C. or, if		when, how, how + adjective *When* shall I come? *How* do you do it? *How big* is it?
6		because	A. Obligatory do, does, did: *Do they* run? *Does it* bite? *Did*n’t *it* hurt? B. Reversal of modal: *Can you* play? *Won’t it* hurt? *Shall I* sit down? C. Tag question: It’s fun, *isn’t it?* It isn’t fun, *is it?*	
7	All other negatives: A. Uncontracted negatives: I can *not* go. He has *not* gone. B. Pronoun-auxiliary or pronoun-copula contraction: I’m *not* coming. He’s *not* here. C. Auxiliary-negative or copula-negative contraction: He was*n’t* going. He has*n’t* been seen. It could*n’t* be mine. They are*n’t* big.			why, what if, how come, how about + gerund *Why* are you crying? *What if* I won’t do it? *How come* he is crying? *How about* coming with me?
8		A. where, when, how, while, whether (or not), till, until, unless, since, before, after, for, as, as + adjective + as, as if, like, that, than I know *where* you are. Don’t come *till* I call. B. Obligatory deletions: I run faster *than* you [run]. I’m *as big as* a man [is big]. It looks *like* a dog [looks]. C. Elliptical deletions (score 0): That’s *why* [I took it]. I know *how* [I can do it]. D. Wh-words + infinitive: I know *how* to do it. I know *where* to go.	A. Reversal of auxiliary have: *Has he* seen you? B. Reversal with two or three auxiliaries: *Has he been* eating? *Could*n’t *he have* waited? *Could he have been* crying? *Would*n’t *he have been* going?	whose, which, which + noun *Whose* car is that? *Which book* do you want?

Appendix C

Example Sentences from Lee

The following tables, providing examples of DSS scoring, are reproduced from Lee (1974).

Table C.1 **Chart 10 from Lee: A hypothetical corpus illustrating a variety of possible DSS scores.**

		Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Pt	Total
1	Boy eat.			-						0	0
2	Boy eat cookie.			-						0	0
3	The boy is eating a cookie.			1						1	2
4	The boys are eating cookies.			2						1	3
5	They ate them.		3,3	2						1	9
6	They didn't eat them.		3,3	6		7				1	20
7	Didn't they eat them?		3,3	6		7		6		1	26
8	Why didn't they eat them?		3,3	6		7		6	7	1	33
9	Why didn't they?		3	inc.		7		6	7	1	24
10	All the cookies were eaten.	3		7						1	11
11	I want to eat some cookies.	3	1	1	2					1	8
12	I want him to eat some cookies.	3	1,2	1	5					1	13
13	I tried to find some cookies.	3	1	2	5					1	12
14	Could you find them?		1,3	6				6		1	17
15	You couldn't find them, could you?		1,3	6		7		6		1	24
16	Nobody knows where to find them.	4	3	2	5		8			1	23
17	Who knows where she keeps them?		2,3	2,2			8		2	1	20
18	I looked but I couldn't find them.		1,1,3	2,6		7	5			1	26
19	I like eating cookies.		1	1	8					1	11
20	Nobody told me that I shouldn't eat them.	4	1,1,3	2,6		7	8			1	33
21	I only ate a few.	7	1	2						1	11
22	Somebody else must have eaten all the rest.	3,3		8						1	15
23	Let's eat some more.	3,3		1	2					1	10
24	Mommy said, "Don't eat those cookies."		3	2,4		4				1	14
25	That isn't what she said.	1	6,2	1,2		5				1	18
26	Him can't have some.	-	-	4		4				0	8
27	What you eating?		1	-				-	2	0	3
28	Her don't gots any.	7	-	-		-				0	7
29	Mommy find out.			-						0	0
30	You want to get spanked?		1	-	7			-		0	8

Total 409

409/30 = 13.63 DSS

Table C.2 **Chart 12 from Lee: Transcript from “C.S.,” a developmentally delayed child of age 3;7.**

		Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Pt	Total
1	You do. (imperative)		1	1						0	2
2	Car go.			-						0	0
3	Dog do.			-						0	0
4	Table move.			-						0	0
5	Mommy clean.			-						0	0
6	I put back.		1	1						0	2
7	I see little.		1	1						0	2
8	I got that. (got/have)	1	1	-						0	2
9	You got paper. (got/have)		1	-						0	1
10	I told you.		1,1	2						1	5
11	Boy take that.	1		-						0	1
12	Dog lie down.			-						0	0
13	Girl ride bike.			-						0	0
14	I no know.		1	-		-				0	1
15	I no take.		1	-		-				0	1
16	I put in here.		1	1						0	2
17	I got two cow. (got/have)	3	1	-						0	4
18	Who broke my chair?		1	2					2	1	6
19	That go in barn.	1		-						0	1
20	He go in bed.		2	-						0	2
21	Who eat my cereal?		1	-					2	0	3
22	Truck no need that.	1		-		-				0	1
23	Table go in here?			-				-		0	0
24	Bed go in here?			-				-		0	0
25	That go in here?	1		-				-		0	1
26	I see doggy on TV.		1	1						0	2
27	I put tea in here. (-/the)		1	1						0	2
28	I get girl and baby. (-/the)		1	1			3			0	5
29	Who sit in my chair?		1	-					2	0	3
30	That little girl ride car.	1		-						0	1
31	He go in he house.		2,-	-						0	2
32	He no go in that.	1	2	-		-				0	3
33	Little baby say, “Who eat my cereal all up?”	3	1	-,-					2	0	6

Total 61

61/33 = 1.85 DSS

Table C.3 **Chart 14 from Lee: Transcript from "A.W.," a normal child of age 2;1.**

		Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Pt	Total
1	I can get fix it.	1	1	-	-					0	2
2	That's broken.	1		1						1	3
3	Baby wakes up.			2						0	2
4	I don't know.		1	4		4				1	10
5	I know.		1	1						1	3
6	She gots coat.		2	-						0	2
7	Her shopping.		-	-						0	0
8	I want to see.		1	1	2					1	5
9	This is talk.	1		-						0	1
10	Get that baby's goes go up. (imperative)	1		1	-,-					0	2
11	It broke it.	1,-		2						0	3
12	This is a baby sock.	1		1						0	2
13	They wake her up.		3,2	-						0	5
14	Her broke a baby's chair.		-	2						0	2
15	It bit you and bite.	1	1	2,-			3			0	7
16	Girl making dinner oatmeal.			-						0	0
17	Look. (imperative)			1						1	2
18	Him have a bath.		-	-						0	0
19	How you open?		1	-				-	5	0	6
20	He have a baseball.		2	-						0	2
21	His shoe will fall off.		2	4						1	7
22	Her sit in chair.		-	-						0	0
23	Her stand up.		-	-						0	0
24	Her can fall off.		-	4						0	4
25	Now he sits up.		2	2						1	5
26	Dolly sit there.			-						0	0
27	Daddy fix it.	1		-						0	1
28	It works.	1		2						1	4
29	It work.	1		-						0	1
30	Fix it. (imperative)	1		1						1	3
31	That's a man.	1		1						1	3
32	Baby fell off.			2						0	2
33	Cow fall off.			-						0	0
34	It fall off.	1		-						0	1
35	Doggie walk.			-						0	0
36	He fall off.		2	-						0	2
37	The girl sitting there.			-						0	0
38	Doggie watching TV.			-						0	0

Total 92

92/38 = 2.42 DSS

Table C.4 **Chart 15 from Lee: Transcript from “A.R.,” a normal child of age 3;7.**

		Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Pt	Total
1	He’s trying to stop everybody.	7	2	1	5					1	16
2	(He’s) he’s putting water on here.		2	1						1	4
3	He’s putting some water in here.	3	2	1						1	7
4	They’re washing dog.		3	2						1	6
5	The dog came out.			2						1	3
6	He jump out.		2	-						0	2
7	She cover her eyes.		2,2	-						0	4
8	(Cause) the soap won’t go in her eyes.		2	4		5				1	12
9	(Because) his shoe came off.		2	2						1	5
10	It’s going by the boy.	1		1						1	3
11	(Took. . .) his shoe fell off.		2	2						1	5
12	(He. . . he. . .) he laughed at the shoe.		2	2						1	5
13	(They) they fell out of his hand.		3,2	2						1	8
14	They didn’t fall.		3	6		7				1	17
15	He’s carrying them with his shirt.		2,3,2	1						1	9
16	(He. . . he’s. . .) he takes it away.	1	2	2						1	6
17	He took the (the) hot-dog.		2	2						1	5
18	(She. . . her. . . she. . . her. . . sher. . .) her toys are falling.		2	2						1	5
19	Those are toys.		3	2						1	6
20	Why are they falling?		3	2				4	7	1	17
21	They fall.		3	1						1	5
22	He eats them.		2,3	2						1	8
23	He eat them.		2,3	-						0	5
24	The dog he (he) barks.		-	2						0	2
25	He (he. . . he. . .) bite them.		2,3	-						0	5
26	He’s putting him here.		2,2	1						1	6
27	Her books fell out of her hand.		2,2	2						1	7
28	She put them in here.		2,3	1						1	7
29	It fell.	1		2						1	4
30	She say, ”Oh, no!”		2	-						0	2
31	He’s trying to take it off.	1	2	1	5					1	10
32	(She. . . she. . .) she’s vacuuming off. (-/him)		2	1						0	3
33	(But) they’re singing now.		3	2						1	6
34	Where’s the sister one? (-/possessive)	3		1				1	2	0	7
35	(But) I want to go.		1	1	2					1	5
36	She drank the soup.		2	2						1	5
37	Her chair fell apart.		2	2						1	5
38	”Who was eating her soup?” (her/my)		-	2					2	0	4
39	He said, ”Where’s my soup?”		2,1	2,1				1	2	1	10
40	(Cause her. . . her. . . cause. . .) her baby bear ate it all.	1,3	2	2						1	9
41	He said, ”Somebody broke it.”	3,1	2	2,2						1	11
42	They fixed it.	1	3	2						1	7
43	They go up. (go/went)		3	-						0	3
44	I don’t know.		1	4		4				1	10
45	She said, ”Get out of my bed.”		2,1	2,1						1	7
46	(She. . . she. . . went. . .) she’s running.		2	1						1	4
47	She went way over here.		2	2						1	5
48	He said, ”Come back.”		2	2,1						1	6
49	”Come back.”			1						1	2
50	She drinked all the soup.	3	2	-						0	5

Total 320

320/50 = 6.40 DSS

Table C.5 **Chart 17 from Lee: Transcript from "N.S.," a normal child of age 2;0.**

		Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Pt	Total
1	Look. (imperative)			1						1	2
2	My coat go? (Where did my coat go?)		1	-				-		0	1
3	I do it.	1	1	-						0	2
4	The baby sit down there.			-						0	0
5	More go. (Some more cars are going)	3		-						0	3
6	Where spoon go?			-				-	2	0	2
7	Baby's eat. (Baby is eating)			-						0	0
8	It fall down.	1		-						0	1
9	It broke.	1		2						1	4
10	Baby fits in that.	1		2						0	3
11	The baby is sleepy.			1						1	2
12	I know.		1	1						1	3
13	(And) that go fall on the baby.	1		-	-					0	1
14	Is this a knife?	1		1				1		1	4
15	Fork fall down.			-						0	0
16	Her crying in there.		-	-						0	0
17	Her crying.		-	-						0	0
18	A knife eat baby. (Baby eats with a knife)			-						0	0

Total 28

28/18 = 1.50 DSS

Table C.6 **Chart 19 from Lee: Transcript from "S.B.," a normal child of age 2;6.**

		Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Pt	Total
1	She washing.		2	-						-	2
2	He wash.		2	-						-	2
3	They riding.		3	-						-	3
4	Look at this. (imperative)	1		1						1	3
5	He fell off. (he/it)		-	2						-	2
6	He look like Roger.		2	-			8			-	10
7	He want baby.		2	-						-	2
8	He was hiding.		2	2						1	5
9	I know where he's hiding.		1,2	1,1			8			1	14
10	He wake up baby.		2	-						-	2
11	She drank it up.	1	2	2						1	6
12	She said, "Sit there." (imperative)		2	2,1						1	6
13	He know.		2	-						-	2
14	They find sleep baby.		3	1	-					-	4
15	She wake up.		2	-						-	2
16	I don't know.		1	4		4				1	10
17	Her fell off.		-	2						-	2
18	That's his chair.	1	2	1						1	5
19	There's one.	3		1						-	4
20	Her bed fit.		2	-						-	2
21	They girls fall down.		-	-						-	0
22	Come on. (imperative)			1						1	2
23	Let's see. (imperative)			1	2					1	4
24	They fit.		3	1						1	5
25	He holding he hand.		2,-	-						-	2
26	Now they hugging.		3	-						-	3
27	They fall.		3	-						-	3
28	They looking me.		3,1	-						-	4
29	(He. . .) he following him.		2,2	-						-	4
30	They can't fit.		3	4		4				1	12
31	They not fit.		3	-		-				-	3
32	Look. (imperative)			1						1	2
										Total	132*

132/32 = 4.12 DSS

*Lee gives the total as 131, and the DSS score as 4.09. This is incorrect, and the correct total and average are shown above.

Appendix D

Example Sentences from Lively

Both of the following sets of sentences are reproduced from the appendices to Lively (1984).

Lively Appendix A

		Indef Pro	Pers Pro	Main Verb	Sec Verb	Neg	Conj	Inter Rev	Wh-Q	Sent Pt	Total
1	I gotta take this thing off.	1	1	-	2					0	4
2	Her gonna talk to Daddy.		-	-	2					0	2
3	Will she help me?		2,1	4				6		1	14
4	Yes, it does.	1		inc.						1	2
5	Here's all the dishes.	3		-						0	3
6	I don't know what to make.		1,6	4	5	4				1	21
7	He climbed the ladder to pick one of these.	3	2,3	2	3					1	14
8	We can get all the stuff out of it.	3,1	3	4						1	12
9	Now you're ready but he has to get his clothes on.		1,2,2	2,2	5		5			1	20
10	Where's those other eyes?	3	3	-				1	2	0	9

Total 101

101/10 = 10.1 DSS

Lively Appendix B

I. Determining sample

   A. *(omitted as irrelevant to SYCORAX)*

II. Sentence Point (grammatically and semantically correct)

   A. Daddy came home. (sentence point)

   B. Carrie brang me some ice cream. (no sentence point)

   C. Mom went to the golf court. (no sentence point)

III. Attempt Mark and Incompletes

   A. Attempt (grammatic, semantic, or pragmatic error)

      1. I maked my bed. (*maked* = attempt mark)
      2. Her is my friend. (*her* = attempt mark)

   B. Incomplete (conversationally appropriate)

      1. Clinician: Are you jumping?
         Child: No, I'm not. (main verb = inc.)
      2. I don't want to. (secondary verb = inc.)
      3. Clinician: Do you know why you're here?
         Child: I don't know why. (conjunction *why* = inc.)

IV. Indefinite Pronouns and Noun Modifiers

   A. No adverbs

      1. Johnny eats more than Bobby. (*more* = indef. pronoun)
      2. Casey wants more cookies. (*more* = noun modifier)
      3. Sally finished last. (*last* = adverb, thus no score)

   B. Numbers

      1. I have fifteen Smurfs. (*fifteen* = 3 as noun modifier)
      2. Carol was third in the race. (*third* = 7 as indef. pronoun)

V. Personal Pronouns (Wh-pronouns vs. Wh-conjunctions vs. Wh-questions)

   A. I know who he is. (*who* = 6 personal pronoun)

   B. I remember where I put them. (*where* = 8 conjunction)

   C. Where are the toys? (*where* = 2 wh-question)

VI. Main Verbs

   A. Have and got

1. I've got three trucks. (*have got* = 7)
2. I got three trucks in my toy box. (*got* = attempt mark)

B. Inflections

1. Smokey is barking. (*is barking* = 1)
2. Smokey was barking. (*was barking* = 2)
3. I want my blanket. (*want* = 1)
4. He wants his blanket. (*wants* = 2)

C. Use of "do"

1. We do see the bikes. (*do see* = 4)
2. We did see the bikes. (*did see* = 6)
3. Do the dishes. (*do* = 1)
4. I did the dishes yesterday. (*did* = 2)

D. Modal auxiliary (inflected vs. uninflected)

1. Alison may go to the store. (*may go* = 4)
2. John might go with her. (*might go* = 6)

E. Must and shall

1. You must finish your dinner. (*must finish* = 7)
2. We shall decide later. (*shall decide* = 7)

F. Perfect tense

1. The kittens have torn the curtains. (*have torn* = 7)
2. The kittens have a new bed. (*have* = 1)
3. The kittens had an old blanket. (*had* = 2)

G. Multiple auxiliaries

1. He has been singing a lot. (*has been singing* = 8)

H. Score form in all relevant categories

1. Didn't we see you yesterday? (*didn't see* = main verb 6, interrog. reversal 6, negative 7)

I. Passives

1. The apple is rotten. (*is* = 1)
2. The cow got milked. (*got milked* = 7)

J. Compound verbs (obligatory vs. optional deletions)

1. They were playing the piano and singing. (*were playing* = 2; *were singing* = 2)
2. The mouse can fit but the cat can't. (*can fit* = 4; *can't* = incomplete in Main Verb category)

VII. Secondary Verbs

A. Absent infinitive marker "to"

1. Make the helicopter go. (infinitive *(to) go* = 5)

B. Five special lexical verbs plus infinitive (2 or 5)

1. I wanna talk. (*talk* = 2)
2. I'm going to talk. (*talk* = 2)
3. I've gotta talk to him. (*talk* = 2)
4. Lemme talk to David. (*talk* = 2)
5. Let's talk now. (*talk* = 2)
6. I want you to talk. (*talk* = 5—different subject)
7. They wanted the girls to talk. (*talk* = 5)
8. Let him talk now. (*talk* = 5)

C. Complementing vs. noncomplementing infinitives

1. He went out to play. (*to play* = 3)
2. They asked me to join. (*to join* = 5)

VIII. Negatives (*this, that,* or *it* + *is* + *not* = 1)

A. This is not mine. (*not* = 1)

B. That's not yours. (*not* = 1)

C. It's not fair. (*not* = 1)

D. Mike was not at home. (*not* = 7)

IX. Conjunctions (Wh-conjunctions: refer to V.)

X. Interrogative Reversals

A. Score both Int. Reversal and Wh-question

1. Where is Ruth? (Int. Rev. = 1; Wh-question = 2)
2. When are we going? (Int. Rev. = 4; Wh-question = 5)

B. Tag questions (6 if correct in all respects)

1. Alicia worked, didn't she? (*didn't she* = 6)
2. Bill isn't home, is he? (*is he* = 6)

XI. Wh-questions (refer to V.)