INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

AN EVALUATION OF A RULE-BASED PARSER OF ENGLISH SENTENCES

Elizabeth A. Scarlett

Thesis

submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Computer Science

March 31, 2000

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

Ottawa, Ontario, Canada

ABSTRACT

Parsers are essential components of many Natural Language Processing applications. In such applications, parsing is the process of analyzing and assigning grammatical types to each word, phrase, and clause in a sentence. A broad-coverage parser attempts to analyze any grammatical sentence of a natural language. A parser depends upon a grammar, which is a set of codified rules of language, to analyze the sentence structure. Often in such systems, the strength of the parser/grammar has a direct effect on the desired results. Achieving good results rests on reliable evaluation methods to determine and eliminate weaknesses in the parser or grammar.

DIPETT (Domain Independent Parser of English Technical Text) is a broad-coverage parser of English technical text that is used primarily in the TANKA (Text Analysis for Knowledge Acquisition) project. The TANKA project seeks to build a model of a technical domain by semi-automatically processing written text that describes the domain. No other source of domain-specific knowledge is available. The accuracy and completeness of a semantic representation generated by TANKA is partly determined by the accuracy of DIPETT's syntactic analysis of the text.

Test suites have long been accepted in NLP because they provide for controlled data that is systematically organized and documented. This thesis investigates the use of test suites for evaluating the performance of parsers of English sentences, and uses two test suites to evaluate DIPETT. The evaluation results were used to make significant improvements to DIPETT, and the test suites were used in part to compare DIPETT's performance to that of several other publicly available large-coverage parsers.

The thesis argues that a test suite for a broad coverage natural language parser must necessarily be systematic, broad in its coverage of phenomena tested, and corpus-like in its coverage of phenomenon interaction. A test suite of example sentences extracted from Quirk *et. al.*'s comprehensive English grammar is proposed, and the results of evaluating DIPETT on that suite are compared with the evaluation results on a publicly available test suite, TSNLP (Test Suites for Natural Language Processing).

TABLE OF CONTENTS

APPENDICES

TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge the help that I have received in preparing this thesis. Grateful thanks to:

- Dr. Stan Szpakowicz, for his guidance, support, and patience, and for the numerous prompt, perceptive, and intelligent comments.
- Terry Copeck for his kind encouragement, and for helping with the parser experiments described in chapter 7.
- Dr. Sylvain Delisle for contributing valuable preliminary comments and suggestions.
- Dr. Ken Barker for providing detailed comments on the final draft.
- Betty Scarlett for proof-reading Appendixes A and B.

I would also like to thank the members of my committee:

- Dr. Franz Oppacher, Carleton University
- Dr. Ken Barker, University of Ottawa
- Dr. Stan Szpakowicz (thesis supervisor), University of Ottawa

DEDICATION

To my parents, who both always stressed the value of an education — my mother, always interested in the syntax of English sentences, and my father, who would have enjoyed Computer Science had such a discipline existed in his day.

AN EVALUATION OF A RULE-BASED PARSER OF ENGLISH SENTENCES

1 Introduction

1.1 Natural Language Processing

Parser Evaluation is an emerging research area within the domain of Natural Language Processing. Natural Language Processing (NLP) is the intersection of many disciplines: artificial intelligence, linguistics, cognitive science, and statistics. In general terms, it is about making computers understand and generate human language. Programming a computer to emulate the language skills of even a very young child is an extremely difficult task, for the knowledge required surpasses the capacity of today's computer systems.

Some potential uses and applications of NLP research are:

- Writers' aids
    - spelling, grammar, and style checkers
    - on-line dictionaries, thesauri, and dictionary access systems
- Information management tools
    - automatic indexing systems
    - text retrieval and information extraction systems, with a natural language component
    - text summarization
- Translators' aids
    - machine translation systems
    - terminology management databases, electronic multilingual dictionaries and thesauri
    - translation memories, specialized workstations
- Natural language front ends for information systems, database systems, computer systems
- Natural language generation modules for larger systems, for example, to synthesize results of information extraction or text summarization

- Speech processing systems
    - speech understanding, recognition, synthesis, communication

1.2 Text Processing in NLP

The need for proficient text processing systems is ever increasing. Two examples of research projects within the School of Information Technology and Engineering are related to the processing of written text.

1.2.1 Text Summarization

The amount of written text in the world, particularly on-line text, has become so great that it is unmanageable without tools for retrieving and filtering. World-Wide Web browsers and search engines often return vast quantities of text, both relevant and irrelevant to the searcher, that are beyond human capacity to scan or absorb. Some of the purposes of information retrieval and text summarization applications are:

- to find appropriate documents on a certain topic,
- to summarize text for certain purposes,
- to produce a shorter version of a text while retaining the main points of the original.

1.2.2 TANKA: Text Analysis for Knowledge Acquisition

Building knowledge bases is essential in the development of AI systems. The development of a new AI system often requires the building of a new knowledge base. Traditionally, knowledge bases are constructed by a cycle of interviewing experts, building a prototype, testing, then re-interviewing the expert. Knowledge acquisition, therefore, remains a difficult, expensive, and labour-intensive task. This is commonly called the *knowledge acquisition bottleneck*. Applications that analyze text for the purpose of knowledge acquisition aim to automatically extract and codify knowledge contained in written texts into a knowledge base.

The TANKA[1] project seeks to build a model of a technical domain by semi-automatically processing written text that describes the domain. No other source of domain-specific knowledge is available.

1.3 Parsers and Parser Evaluation

Parsers are essential components of many NLP applications. In such applications, parsing is the process of analyzing and assigning grammatical types to each word, phrase, and clause in a sentence. For example, a parser recognizes noun phrases, verb phrases, and relative clauses as well as parts of speech. A *broad-coverage parser* attempts to analyze *any* grammatical sentence of a natural language. A parser depends upon a *grammar*, which is codified rules of language, to analyze the sentence structure. The parser uses the grammar to find analyses; the process as a whole may be seen as a form of deduction. The output of a parser/grammar is an analytical representation of a sentence called a parse *tree*. Parser output provides more syntactic information than flat part-of-speech annotations, but does not generate a full semantic representation of a sentence.

Often in such systems, the strength of the parser/grammar has a direct effect on the desired results. Achieving good results rests on reliable evaluation methods to determine and improve weaknesses in the parser and grammar.

1.3.1 The Domain Independent Parser of English Technical Text

DIPETT[2] is a broad-coverage parser of English technical text that was developed for a Ph.D. research project (Delisle, 1994), and is used primarily in the TANKA project. Broad-coverage parsing is required to parse large-scale technical text. The accuracy and completeness of a semantic representation generated by TANKA is partly determined by the accuracy of DIPETT's syntactic analysis of the text. DIPETT is a linguistic *theory-neutral* parser (it does not presuppose any particular linguistic theory of language) with a broad surface-syntactic coverage of English. The core of DIPETT's grammar is based on Quirk *et*

[1] Text Analysis for Knowledge Acquisition

[2] Domain Independent Parser of English Technical Text

al. (1985), a standard academic grammar of English, and Winograd (1983). DIPETT is implemented as a logic grammar (Prolog) and is particularly rich in "lookahead" and heuristics for efficient parsing of complex syntactic structures.

1.4 Goals of the Thesis

The goal of this thesis was to investigate the use of test suites for evaluating the performance of parsers of English sentences, and to use such a test suite to evaluate our in-house parser, DIPETT. The evaluation results were used the to make improvements to DIPETT, and the test suites were used in part to compare DIPETT's performance to that of other publicly available large-coverage parsers.

The thesis shows that a test suite for a broad coverage natural language parser must necessarily be systematic, broad in its coverage of phenomena tested, and corpus-like[3] in its coverage of phenomenon interaction.

I propose a test suite of example sentences extracted from Quirk *et al.*'s comprehensive English grammar (Quirk *et al* 1973, 1985), and compare the results of evaluating DIPETT on that suite with the evaluation results on another publicly available test suite, TSNLP[4] (Lehmann *et al.*, 1996b).

1.5 Organization of the Thesis

Chapter 2 of the thesis gives an overview of general parser evaluation literature. Various methods of parser evaluation are discussed, including diagnostic and progress evaluation.

Chapter 3 gives a summary of the literature from the TSNLP project. One of the tools commonly used for diagnostic evaluation is a test suite; the TSNLP project was concerned with the design and use of test suites in NLP. The design of the TSNLP test data and their annotation schema is described.

[3] Corpora comprise arbitrary (but relevant) combinations of phenomena.

[4] Test Suites for Natural Language Processing

Chapter 4 reviews the existing test methods and evaluations of DIPETT. A proposal for a broad coverage test suite for DIPETT is presented, as well as a proposal for a diagnostic and progress test system for DIPETT.

Chapter 5 presents a diagnostic evaluation of DIPETT on the test suite described in chapter 4, and on the TSNLP sentences, and compares the resulting evaluations.

Chapter 6 shows how the evaluation results can be used to make improvements to DIPETT. Some improvements are described, and their effect on DIPETT's performance is analyzed.

Chapter 7 provides a brief comparison of DIPETT with three other publicly available parsers.

Chapter 8 gives a summary of the thesis, discusses problems, and presents future work to be done in evaluating and optimizing DIPETT.

2 NLP and Parser Evaluation Literature

This chapter presents a general survey of the literature and current research projects relating to parser evaluation. The following are presented:

1. The Sparck Jones and Galliers review, which mentions some long running research projects.
2. The EAGLES[5] project and the Human Language Technology (HLT) survey, which introduce NLP evaluation terminology.
3. A brief summary of parser evaluation metrics and research this past decade.
4. Short descriptions of some current large parser evaluation projects.
5. The most recent proposals for solving some of the problems outlined in part 3.

Research in parser evaluation has mainly been conducted and published through conference workshops. Workshop proceedings and papers published prior to 1995 are often difficult to acquire. However, copies of later papers are often available through the Internet, and some of these and other publications provide good reviews of early research in this field.

2.1 "Evaluating Natural Language Processing Systems: An Analysis and Review"

Sparck Jones and Galliers (1996) provide an outline of early efforts made to evaluate NLP systems. The material in this book is not specific to parser evaluation; indeed, natural language parsers are barely mentioned. The book is concerned with *performance evaluation* of NLP systems as a whole, not just the performance of the parser/grammar. The book provides a comprehensive review of research activity relating to NLP evaluation. It ranges across a broad spectrum of NLP applications, including generic systems, machine translation, message understanding, speech understanding, speech recognition. This book is generally too broad to be very useful for this thesis, although it does describe the early parser/grammar evaluation (Parseval) workshops and the EAGLES and TSNLP projects (as

[5] Expert Advisory Group on Language Engineering Standards

they existed before 1995). The final state and contribution of these research initiatives will be described below, as will the concept of performance evaluation. A review of the Sparck Jones book was published by Walter (1998).

2.2 Evaluation Guidelines

The EAGLES project subgroup on evaluation suggested that three purposes of evaluation could be distinguished (Balkan *et al.*, 1994c; Sparck Jones & Galliers, 1996):

- *Diagnostic evaluation* aims at localizing deficiencies in a system.
- *Progress evaluation* compares successive stages of development of a system over time.
- *Adequacy evaluation* determines whether a system meets some specified requirement.

The EAGLES report also provides a survey of *glass-box* and *black-box* test techniques for testing software (*EAGLES*, 1995):

- *Black-box test methods* do not use explicit knowledge of the system internals. Black-box testing focuses on testing of functional requirements and so tests for correct outputs.
- *Glass-box test methods* use knowledge of the system internals to guide the selection of test data and is therefore the testing of known software paths.

2.3 Survey of the State of the Art in Human Language Technology

The HLT Survey – Survey of the State of the Art in Human Language Technology (Cole *et al.*, 1996) presents a useful and descriptive introduction to NLP evaluation terminology. This survey covers a broad spectrum of NLP products and disciplines: text analysis, machine translation, broad-coverage natural language parsers, speech processing, and character recognition. The authors distinguish three different goals and three classes of evaluation.

2.3.1 Adequacy Evaluation

Adequacy evaluation determines whether an NLP system will suit the needs of a given purpose: will it meet the user requirements, how well, at what cost. It provides users and

application developers information to determine whether an NLP system is adequate for their particular task, and if so, whether it is more suited than others. Both diagnostic and performance evaluation techniques may be used, but are unlikely themselves to be sufficient. Described as following the "Consumer Reports paradigm", adequacy evaluation provides comparative information of parsers but does not necessarily identify the best parser, thus allowing the user to make an informed choice.

2.3.2 Diagnostic Evaluation

Diagnostic evaluation tests system performance with respect to the space of possible inputs. In NLP application areas where coverage is important, particularly systems with explicit grammars, large *test suites* are commonly used for diagnostic evaluation. The purpose of a test suite is to enumerate linguistic phenomena in the input domain and their most likely and/or important combinations. A test suite may include invalid as well as valid inputs. Test suites allow automated regression testing to ensure that system changes have the intended effect and no others. Possible limitations of diagnostic evaluation are (1) such test suites may not reflect the distribution of linguistic phenomena in actual application domains, and (2) coverage of any particular linguistic phenomena is not in itself indicative of fitness of an application to a user's purpose.

2.3.3 Performance Evaluation

Performance evaluation is a *quantitative* measurement of system performance in one or more specific areas. We distinguish:

Criterion: What it is we are interested in evaluating — precision, speed, error rate.

Measure: Which specific property of system performance we report in an attempt to measure the chosen criterion — ratio of successes to total outputs, seconds to process, percent incorrect.

Method: How the appropriate value for a given measure and a given system may be determined.

Such metrics have traditionally been applied to quantify the performance of information retrieval systems, although the concepts have been adopted and are now being used in NLP evaluation as a whole. Much of the discussion in parser evaluation literature has sought to determine appropriate *measures* for evaluating and comparing output analyses of parsing systems.

2.3.4 Evaluation of Broad-Coverage Natural Language Parsers

According to Ezra Black (Cole *et al.*, 1996; chapter 13.4) objective evaluation of natural language parsers essentially was not practiced at all Before 1991. In 1991 the Parseval system for syntactically evaluating broad-coverage English-language parsers was introduced, and has become widely adopted in spite of several limitations and drawbacks (which will be discussed below).

Parser developers now recognize the necessity of rigorous testing for effective parsing-system development. Comparative evaluation of broad-coverage parsers (of a given language) in terms of a common analysis has been discussed in the literature, but may never be possible or ultimately even desirable. Diagnostic tests that determine accurate and rigorous performance statistics on individual systems, rather than coarser-grained scores obtained from comparing systems with others, may be a preferable objective.

2.4 Reviews of Existing Methods: Evaluation of Parsing Systems Workshop

Carroll *et al.* (1998) and Srinivas *et. al* (1998) have published recent papers that provide an excellent review of existing methods of parser evaluation and research in parser evaluation since 1991. Both of these papers were presented at The Evaluation of Parsing Systems Workshop of the First International Conference on Language Resources and Evaluation at the University of Granada, 1998.

Carroll *et al.* (1998) organize parser evaluation methods into three categories: no corpus, unannotated corpus, and annotated corpus methods. Much of the same material was earlier

published by the SPARKLE[6] project group (Carroll *et al.*, 1997). Srinivas *et al.* (1998) organize parser evaluation methods into three different categories: extrinsic, intrinsic, and comparative methods. Since all three sources present much the same information, their summaries are combined here.

A *corpus* is a collection of sentences that may be annotated with information such as part-of-speech tags for each word. The corpus annotation is known as the *gold standard.* The accuracy of annotated corpus evaluation metrics depends upon the accuracy of the gold standard.

2.4.1 Intrinsic Evaluation

2.4.1.1 Test Suite based evaluation (No corpus)

Test suite based evaluation is the same as diagnostic evaluation defined above. The traditional approach to evaluating a parser's grammatical coverage consisted of listing the constructions covered and those not covered by a given grammar or parser (for example, see Grover *et al.*, 1993). A test suite, usually a list of sentences maintained as a database, is used to track improvements and verify consistency between successive generations of grammar development (progress evaluation). Test suites may also be used to pinpoint faults in a grammar (diagnostic evaluation). The advantages are that this is relatively easy, requires no corpus, and parse results provide a direction for improving the system.

The disadvantages are that performance on a test suite does not provide information about coverage of English sentences in real text. For instance, rare and marginal constructions are often not covered by a grammar or tested through the test sentences. Also, coverage of various constructions does not guarantee coverage of their interaction in combination with others.

[6] Shallow Parsing and Knowledge Extraction for Language Engineering

2.4.1.2 Coverage (Unannotated corpus)

Conventional grammars can be seen as collections of rules that precisely specify possible constituent structures. By defining the set of legal structures, the grammar also defines the class of legal sentences.

A useful first step in evaluation is to determine the percentage of examples from a given corpus for which the parser/grammar is able to assign at least one complete (not fragmentary) analysis (Briscoe & Carroll, 1995). *Coverage* is inversely proportional to the number of sentences for which no full parse tree is generated. This calculation does not require corpus annotation and can be made for a large corpus. The performance measure calculated is very weak because there is no guarantee that any of the analyses (parse trees) found are completely correct. A parser may generate a full parse tree that is incorrect because more than one representation of a sentence may be possible. Prepositional phrase mis-attachment is an example. So there may be incorrect yet full parse trees included in the result and these can be found only by manually checking the output, a labourious process. In addition, more permissive grammars will score more highly using this measure. The grammar `S -> W*` (a sentence can be composed of any sequence of words) would do very well on this measure.

2.4.1.3 Average Parse Base (Unannotated corpus)

It is possible for a sentence to have more than one correct analysis generated by a grammar; analyses that may or may not be semantically and pragmatically reasonable. Very often, a grammar that provides good coverage does so at the expense of allowing a large number of spurious parses. This problem is known as *over-generation.*

The IBM group (Black *et al.*, 1993) suggested calculating the *parse base* (PB). They define the PB as the geometric mean of the number of analyses (parse trees generated) divided by the number of input tokens (words) in each sentence parsed. One can, using this measure, predict the average number of parses that will be generated for a sentence of n words. The IBM group describe a problem with PB, that is, it assumes that numbers of parses grow linearly with sentence length while in practice it appears that the relationship is "more nearly

exponential" (Carroll *et al.*, 1998). PB therefore overestimates the expected number of parses for short sentences and underestimates for long sentences.

To address this problem, Briscoe and Carroll (1995) provide a revised measure of ambiguity rate, the *average parse base* (APB). This is defined as "the geometric mean over all sentences in the corpus of $\sqrt[n]{p}$ where n is the number of words in a sentence, and p, the number of parses for that sentence". A sentence of length n drawn from the corpus and analyzed with that grammar/parser would be expected to give APB^n parses.

APB can be computed for large corpora and provides an estimate of the degree of ambiguity in a grammar. This is a useful measure to keep track of during grammar development. Calculating APB for different versions of a grammar tested on the same data ensures that the coverage is increasing but the ambiguity is not. Disadvantages of the APB measure are: a low-coverage unambiguous grammar would do very well on this measure alone. APB should be used in conjunction with an independent measure of coverage. The main disadvantage though, is that it is not possible to meaningfully compare performance of different parsers on different data, as the inherent ambiguity in the data interacts with the ambiguity of the grammar.

2.4.1.4 Entropy (Unannotated corpus)

Statistical parsing methods attempt to solve the over-generation problem outlined above by allowing the parsing system to assign scores to analyses. A common way to assign weights is to use a *probabilistic grammar*. A probabilistic grammar (Brew & Moens, 1998) assigns a set of legal structures to each word sequence of the language, and gives a probability to each such structure. The probability of a word sequence is the sum of the individual probabilities of all its structures. Word sequences that have no legal analysis according to the grammar have zero probability. In many cases such grammars are induced from an annotated corpus data rather than coded by people.

A probabilistic grammar can be used to calculate the entropy (randomness) of an unannotated corpus. This yields a measure of how well induction of a grammar has captured

regularities in the corpus through minimizing unpredictability and ambiguity. This measure is specific to a given probabilistic grammar and corpus, but the approach can be used to compare the effectiveness of different grammars.

The main advantage of the entropy measure is that it allows meaningful comparison of different probabilistic grammars of the same corpus. The disadvantages are that it is applicable only to probabilistic grammars, and only provides at best a weak measure. That is, a low ambiguity score on the corpus does not mean that the probabilistic grammar derived from it will necessarily be accurate.

2.4.1.5 Part-of-speech Assignment Accuracy (Annotated corpus)

A part-of-speech tagger is an NLP system or subsystem whose purpose is to identify or assign the correct lexical syntactic category to a word in a sentence. The *accuracy* with which a tagger or parser/grammar performs this function has been measured in a variety of ways: ratio of correct tags per word, possibly subdivided into ambiguous and unambiguous and/or known and unknown words. An advantage of using this measure is that there is a considerable amount of manually corrected tagged material to use as test corpora (for English). The disadvantages are that this measure cannot be used to evaluate broad-coverage parsers that use pre-tagged text as input, and many of them do. Also, it provides only a partial measure of the accuracy of syntactic analysis. This measure is used mostly to evaluate taggers and lexical parsers.

2.4.1.6 Zero Crossing Brackets (Annotated corpus)

Parsers are either *constituency-based* or *dependency-based* systems. Dependency-based parsers are described below. Constituency-based systems represent sentences and phrases as a tree structure that is generated according to the grammar's *phrase structure rules*. The phrase structure rules embody the grammar's definition of language syntax. *Constituency* is the relationship between a whole syntactical element and its parts, as defined by the phrase structure rules.

A common sort of parse tree representation identifies the constituent parts of the sentence using a bracket notation. Such a bracket notation can easily identify nested constituents, and alternative representations for the same sentence. For example:

[[the dog] [saw [the duck]] [in [the pond]]] [[the dog] [saw [the duck [in [the pond]]]]]

A phrase boundary can be represented by an interval $[i, j]$ where i is the index of the first word in the phrase and j is the index of the last word in the phrase. i and j have the same value for a single word phrase.

In an attempt to directly measure parsing accuracy, the IBM group defines *structural consistency*. This is the percentage of sentences for a test corpus which receive at least one complete analysis, of which one has no *crossing brackets* when compared with the gold standard. The crossing brackets measure is essentially the number of constituents in the analysis that are inconsistent with the gold standard. Conversely, one could count the constituents that exactly match the gold standard structure. A phrase boundary in a parse tree $[i, j]$ and another in the gold standard $[i', j']$ have crossing brackets if $i < i' <= j < j'$ that is, if the phrase boundaries overlap and neither phrase is properly contained in the other.

The structural consistency measure is somewhat better than simply measuring coverage, but requires an annotated corpus. The drawbacks are that it favours systems which yield relatively flat parse trees — there will be no crossing brackets if the parser uses the grammar `S -> W*`.

2.4.1.7 Best-first/Ranked Constituency (Annotated corpus)

Briscoe and Carroll (1993) measure the accuracy of probabilistic parsing systems as the percentage of highest-ranked analyses output by a parser/grammar that are identical to the gold standard. This measure may be extended so that the top n analyses are compared, and scoring is relative to their ranking. This measure gives a meaningful score of how often a parser would deliver an appropriate analysis. However, it is dependent on having access to an accurate annotated corpus that is fully compatible with the parser output. Again, measures that are based only on the absence of crossing errors on the sentence level are not usable for

parsing systems that supply fragmentary parses (partial bracketing) since a sparse bracketing improves the score.

2.4.1.8 Tree Similarity (Annotated corpus)

Tree similarity measures of various types have been defined, for example, the ratio of rules correctly used in a derivation over the rules used in the gold standard derivation.

This measure is more fine-grained than full parse tree identity. A grammar/parser may never produce fully correct analyses yet may consistently produce analyses that are very similar to the correct one. This measure is also more tolerant of errors in the annotated test data. However, it is also a weaker measure than full identity and of course, it still needs a detailed and compatible annotated test corpus.

2.4.2 Extrinsic Methods

Extrinsic evaluation is possible when a parsing system is embedded in an application and the parsing system's contribution to the overall performance of the application can be measured. The objective of this type of evaluation is *adequacy evaluation* of a parser. Extrinsic evaluation can also be used as an indirect method for comparing parsing systems even if they produce different representations for their outputs, as long as the output can be converted into a form which may be used by the application.

2.4.3 Comparative Methods

The objective of comparative evaluation methods is to directly compare the performance of two or more parsing systems that use different grammar formalisms or different statistical models. Comparative evaluation helps in identifying the relative strengths and weaknesses of the systems, and may suggest possibilities of combining different approaches. However, such an evaluation scheme requires a metric that is insensitive to the representational differences in the output produced by different parsers. To achieve this, the metric may have to abstract away from individual representations in order to be able to directly compare them. A drawback of this approach is that the strengths of representation of certain parsers might be lost completely because of the abstraction process.

2.4.3.1 Parseval/Penn Treebank (Annotated corpus)

An early parser evaluation scheme, designed for comparative evaluation of parsing systems, is the Parseval/Penn Treebank system (Harrison *et al.*, 1991; Marcus *et al.*, 1993). The Parseval system has gained wide usage, in spite of several limitations and drawbacks, possibly because no workable alternatives have been developed. The Parseval scheme provides an evaluation metric that made it possible to quantify and compare the performance of different parsers – a previously impossible task. The Parseval evaluation procedure compares parser output to the annotated analyses given in a *treebank*. A treebank is an annotated set of sentences, similar to a gold standard, except that the only annotation required for the treebank is phrase bracketings.

The Parseval scheme uses bracketing information from the treebank representation of a sentence and the parse analysis of a sentence to compute three figures (Srinivas *et al.*, 1998):

1. crossing brackets, as described above
2. recall, a ratio of the number of correct brackets in the system parse to the total number of brackets in the treebank parse
3. precision, a ratio of the number of correct brackets in the system's parse to the total number of brackets in the system's parse.

There have been several objections to the limitations of this scheme:

- The original published description and the evaluation software are specific to the English language.
- A significant limitation of the Parseval metric is that it was not designed to evaluate parsers that produce partial (fragmentary) parses. Further, it can only be applied to parse trees generated by constituency-based parsers.
- It is not fine-grained enough to evaluate parsers with respect to specific syntactic phenomena.

- Sparck Jones and Galliers (1996) write that there was little consensus on sentence structure across candidate analyses for the same sentence. That is, there was little agreement on how the treebank parses should be represented.
- Carroll *et al.* (1999) states that the treebank structures have been criticized as being 'flat' (contain relatively few brackets) and so, crossing-bracket scores will tend to be low.
- Srinivas *et al.* (1995) observed that the precision measure penalizes a parser for inserting extra, possibly correct brackets, if the treebank annotation is skeletal.
- Lin (1995) observed that the crossing-brackets measure penalizes mis-attachments more than once. A shallow parse, containing less syntactic information, scores better on the Parseval metric.

2.4.3.2 Dependency Structure-based Scheme (Annotated corpus)

Dependency-based parsers produce a set of dependency linkages between the words of a sentence. The fundamental assumption of dependency grammar theory is that syntactic structure consists of lexical nodes (representing words) and binary relations (dependencies) linking them.

In contrast to phrase structure (constituency based) representations, dependency representations have no phrase structure categories, that is, there are no phrasal nodes. A *dependency* exists between two words if one word is sensitive to or conditioned upon a property of the other, as defined by some grammar rule. Dependency relations are binary, directed, and may be typed in order to distinguish different kinds of dependency.

To overcome the Parseval/Penn Treebank grammar/treebank bracket mismatch problems, Lin (1995) proposes evaluation based on dependency structure, in which phrase structure analyses from parser and treebank are both automatically converted into sets of dependency relationships. The dependency relationships, rather than phrase boundaries, are then used in the comparison. Lin states that the advantages of dependency-based evaluations are: inconsequential differences (in bracketing) are ignored; parsers may be selectively evaluated

with respect to any given type of syntactic phenomena; and errors in parser output may be pin-pointed precisely.

Hogenhout and Matsumoto (1996) state that Lin's alternative discussed above is difficult to apply. They suggest a more detailed evaluation method using constituent boundaries, and suggest a new set of measures (all related to counting of brackets) which give more information than the current measures. Their proposal is not addressed, however, in either Carroll *et al.* (1998) or Srinivas *et al.* (1998) (described above).

Carroll *et al.* (1998) does mention Atwell's (1996) objection to Lin's approach, that transforming standard constituency-based analyses into a dependency-based representation would lose certain kinds of grammatical information.

2.4.3.3 Relation Model

Srinivas *et al.* (1996) propose an alternative comparative parser evaluation method which overcomes some of the limitations of the Parseval scheme, and can be used to evaluate both full and partial parses. They review this proposal in Srinivas *et al.* (1998). The objective of the relation model is to compare parsers irrespective of their output representation. Srinivas *et al.* suggest a normalized representation of a parser's output, using chunks and dependency links.

According to the literature (Carroll *et al.* , 1997) a *chunk* is a unit of adjacent word tokens, from text, which are linked by dependency links. Each identified chunk is labeled as to its category — adjectival, adverbial, finite verb, nominal, prepositional, participial. Chunk categories are more granular than the syntactic phrasal categories typically used in constituency-based syntax.

Srinivas *et al.* (1998) propose to normalize the output of parsers which produce highly structured analyses (for example, constituency-based parsers) by flattening phrasal constituents to chunks. Then, the accuracy of a structure can be measured as the accuracy of the chunks and the relations between chunks (expressed as dependency links between the

heads). Fragmentary parses are penalized because unattached constituents are missing dependency links.

2.5 Some Recent NLP Evaluation Projects

2.5.1 SPARKLE – Shallow Parsing and Knowledge Extraction for Language Engineering

SPARKLE (Carroll *et al.*, 1997) is a large research consortium with participation by researchers based in Italy, England, France, and Germany. The goal of the SPARKLE project is to develop flexible, multilingual tools for semi-automatic acquisition of *linguistic* knowledge from text. The goals of the SPARKLE project have been and are to be realized in stages. First, to develop shallow parsers able to produce phrasal-level syntactic analysis of naturally occurring free text (English, French, German, and Italian). Second, to develop a lexical acquisition system capable of learning, from free text, the knowledge of words that is essential for NLP applications. The SPARKLE acquisition system is intended to process the output of the shallow parsers to extract lexical knowledge about semantic classes of predicates, subcategorization, and argument structure for the language of focus. The third goal is to verify and validate the results and apply them to an industrial pilot project. SPARKLE is an ongoing project, 1996 - present, and provides funding for parser evaluation research.

2.5.2 EAGLES - The Expert Advisory Group on Language Engineering Standards

The purpose of the EAGLES project (1993-1996) was

- to accelerate the provision of standards for very large-scale language resources such as text corpora and computational lexicons
- to develop means of manipulating such knowledge; and
- to develop means of assessing and evaluating natural language resources, tools, and products.

The ISO 9126 Standard sets out a general framework for designing an evaluation. The EAGLES group, together with the TEMAA[7] project (*TEMAA*, 1997) seeks to apply this general framework to perform an adequacy evaluation of products in the areas of writers' aids (spelling and grammar checkers are two examples) and translators' aids. This has led to augmentation of the ISO 9126 Standard (*EAGLES*, 1995).

The lexicon/syntax interest group of EAGLES (Sanfilippo *et al.*, 1996) has developed a specification of grammatical dependencies for verbal arguments. A tailored version of this standard became the basis for the most recent proposal for a parser evaluation system (Carroll *et al.*, 1998, described below).

EAGLES-II (1997-1998) held two intensive workshops to consolidate, extend and disseminate the EAGLES work.

2.5.3 TEMAA - A Testbed Study of Evaluation Methodologies: Authoring Aids

The TEMAA project objective (*TEMAA*, 1997) was to develop a conceptual framework for an evaluation methodology for NL products. The TEMAA project focused on adequacy evaluation, specifically the evaluation of spelling checkers and of grammar checkers, and used as input the work of the EAGLES Evaluation working group. An important achievement of the TEMAA project was the design of a formal evaluation framework, the parameterizable test bed (PTB). The PTB is implemented as a software program that outputs an evaluation of the NL product.

2.5.4 TSNLP - Test Suites for Natural Language Processing

The TSNLP — Test Suites for Natural Language Processing project (Lehmann *et al.*, 1996b) started in December 1993 and ended in October 1995. The project was concerned with the design and use of test suites in NLP. Because the TSNLP test suite was used extensively for this thesis, the TSNLP literature is reviewed in some detail in the next chapter.

[7] Testbed Study of Evaluation Methodologies: Authoring Aids

2.5.5 DiET- Diagnostic and Evaluation Tools for Natural Language Applications

The TSNLP successor DiET (Diagnostic and Evaluation Tools for Natural Language Applications) is expected to deliver greatly improved access tools and corrected TSNLP test data (Netter *et. al.*, 1998).

2.6 A Current Proposal for Evaluation Systems

Measuring parser accuracy has proved to be a difficult problem, and a number of approaches have been proposed in the literature. Constituency-based evaluation methods have serious drawbacks.

Carroll *et al.* (1998) describe and justify a new dependency-based technique which overcomes some of the shortcomings of previous proposals. The *grammatical relation* scheme (Carroll *et al.*, 1997) is only superficially similar to the scheme proposed by Lin (1995). The grammatical relation scheme is built upon modifications and refinements to the EAGLES subcategorization standard (Sanfilippo *et al.*, 1996). A grammatical relation (GR) specifies the syntactic dependency that holds between a head and a dependent. GRs are organized hierarchically. A number of GRs have been defined.

Parser evaluation using the grammatical analysis scheme is based on calculating:

1. *Recall* — the number of GRs returned by the parser that match GRs in the annotated sentence divided by the total number of GRs in the annotated sentence.
2. *Precision* — the number of GRs returned by the parser that match GRs in the annotated sentence divided by the total number of GRs returned by the parser for that sentence.
3. *F-score* — a measure combining precision and recall into a single figure.

Carroll *et al.* (1999) describe a new 10k-word corpus of naturally-occurring English sentences, annotated to this standard and show an example of its use in evaluating a robust parsing system. This new evaluation scheme has been used in the SPARKLE project to evaluate their four (multilingual) broad-coverage shallow parsers.

3 Test Suites for Natural Language Processing

3.1 Parser Evaluation and DIPETT

As shown in chapter 2, some parser evaluation methods are used for the purpose of directing and monitoring the development of parser/grammar systems, while others are appropriate for comparing different systems.

The primary goal of this thesis is to evaluate our in-house parser, DIPETT, for the former purpose — diagnostic and progress evaluation. That is, the goal is to evaluate DIPETT to extend its coverage, improve the quality of the parse tree output, and possibly find and reduce processing inefficiencies.

It is appropriate to use a *test suite* (intrinsic evaluation, test suite based evaluation) to achieve these goals. As the TSNLP test suite was used extensively for this purpose, this chapter provides a review of the design and development of the test suite.

3.2 TSNLP – Test Suites for Natural Language Processing

3.2.1 TSNLP Background

The TSNLP (Test Suites for Natural Language Processing) project was started in 1993 and finished in 1996. The purpose of the project was to investigate all aspects of the construction, maintenance, and application of systematic test suites as diagnostic and evaluation tools for NLP applications. The TSNLP project produced much insight into the nature of test suite design and delivered three large, publicly available, multi-purpose test suites (English, French, and German). TSNLP was a huge consortium-based project funded by LRE[8]. The consortium participants were the University of Essex (England), DFKI[9] (Germany), ISSCO[10] (Switzerland), and Aerospatiale[11] (France). The consortium produced twelve conference and

[8] The Linguistic Research and Engineering Programme of the European Commission (DG XIII).

[9] Deutsches Forschungszentrum für Künstlich Intelligenz GmbH. Saarbrücken, Germany.

[10] The Istituto dalle Molle per gli studii semantici e cognitivi. Geneva, Switzerland.

[11] The Common Research Centre of Aerospatiale. Suresnes, France.

book articles and ten project reports (work package deliverables). It is not possible (nor is it necessary) to review all of the TSNLP material in depth for this thesis.

The TSNLP project (Balkan *et al.*, 1994c) defines a test suite for NLP as "a more or less systematic collection of specially constructed linguistic expressions (test items, phrases or sentences), perhaps with associated annotations and descriptions." Most of the test suites that existed in 1994 had been written for specific existing systems, or were simply collections of "interesting" sentences.

The TSNLP project had a number of goals:

1. To provide guidelines for test suite construction.
2. To develop an annotation schema that would be neutral to linguistic theory, evaluation type, and application type.
3. To construct substantial multilingual test fragments applicable to a broad range of NLP application types.
4. To develop a linguistic database to store, access, and manipulate test data items and their corresponding annotations and analysis.
5. To investigate (semi-)automated test suite construction tools and methods.

By 1996, the TSNLP project had delivered a large broad-coverage test suite. The characteristics of the TSNLP test suite are:

- It supports various types of testing and evaluation (diagnostic, progress, adequacy) and is relevant to a number of applications (including parsers and grammar checkers).
- It is useful to both application developers and applications users.
- It is multilingual: Test data are available in English, French, and German, and have been produced in parallel in that they test the same range of phenomena. The test data can be extended to other languages.

- It is reusable: The test items are stored in a relational database, where test items created for different purposes are associated with detailed annotations, and from which application- and purpose-specific test items can be extracted.

3.2.2 Study of Existing Test Suites

During the first phase of the TSNLP project, a study of existing publicly available test suites was performed (Estival *et al.*, 1994a). The purpose of the study was to identify relevant characteristics for the description of test suites and to get an overview of the types of test suites that existed. Nine test suites chosen and tested differed in a number of properties:

- Source language: English, German, French.
- Size: 346 to 3200 test items.
- State of development: none claimed to be complete.
- Purpose: diagnostic, in all cases, although some were also used for the purpose of adequacy evaluation.
- Intended target: four for parsers, four for MT systems, one for database query interface.
- Intended user: six for system developers, three for users or external evaluators.
- System state: six for systems under development, three for commercial systems

Generally, it was found that the existing test suites were only somewhat systematic. Although they were primarily concerned with coverage of syntactic phenomena, seven of the nine had been designed in an ad hoc manner. The general shortcomings found were:

- Lack of coverage of phenomena other than syntactic phenomena, for example, morphology (word formation), semantics (sentence meaning), and extra-linguistic items (punctuation, abbreviations, acronyms, and formatting).
- Lack of systematic testing of ill-formed constructions.
- Lack of systematic testing of the co-occurrence of different constructions.
- Lack of documentation and annotation.

- Most were too system-specific to be reusable.
- Difficult to interpret results.

The main conclusion of this report was that the design of test suites could be improved to make them more useful, reusable, portable, and maintainable.

3.2.3 Test suites vs. Text corpora

An important issue in test suite design was to consider what exactly are the differences between a test suite and a corpus (Balkan *et al.*, 1994a and 1994b). In general terms, test suites are constructed of systematically chosen test items, while a corpus consists of selections of naturally occurring text. For certain purposes, test suites provide a more direct tool. Some of the differences between test suites and corpora are:

Control over test data

Test suites allow for systematic construction of test data focused upon specific phenomena. The phenomena may be tested alone or in combination with others. Corpora comprise arbitrary (but relevant) combinations of phenomena. Test suites allow for greater focus and more control of test data than do corpora.

Systematic coverage of phenomena

Tests suites may provide test items that systematically test variations over a specific phenomenon. Such variations occur only by accident in a corpus, as a corpus provides only a representational sample.

Non-redundant representation of phenomena

Systematically constructed test suites test a phenomenon only once. The same phenomena may recur several times in a corpus, as a corpus is meant to show distribution and frequency.

Negative examples

Testing negative examples may be useful in diagnostic evaluation of any NLP application, and are particularly necessary for testing grammar-checking applications. Ungrammatical items do not occur (systematically) naturally in corpora.

Annotation of test items

Items in test suites and in corpora may be annotated. Test suites are more likely to be annotated with items specifically relevant to diagnostic evaluation. Annotations may be specific to a certain application or may be more general. For example, annotations showing the structure of the test items might classify the phenomenon tested, the grammaticality and centrality of the phenomenon, and might show morpho-syntactic tagging of the test item, and possibly an abstract dependency-oriented analysis of the test item.

3.2.4 Methodology of Test Suite Design

Following completion of the survey of existing test suites and an analysis of diagnostic and evaluation requirements of developers and users, the TSNLP group devised a method for the construction of *core test data* (Balkan *et al.*, 1994a, 1996). The TSNLP group define core test data as test items such as sentences. The test items reflect central language phenomena applicable to a wide range of NLP applications, including parsers and grammar checkers (Lehmann *et al.*, 1996b). The TSNLP method is designed to optimize *control over test data*, *progressivity*, and *systematicity* (Balkan *et al.*, 1996) in order to construct a test suite that is adequately broad-coverage, multi-purpose, multi-user, multi-lingual, reusable, and has the advantages over corpora as described above.

Control over test data

TSNLP restricts the vocabulary in size and domain. Ambiguous words are avoided and are only included when that ambiguity is explicitly tested. The interaction of phenomena is controlled by keeping the most basic test items deliberately simple. A number of guidelines were set forth for this purpose (see Estival *et al.*, 1995). Phenomena are tested in isolation and in controlled combinations.

Progressivity

Progressivity is the principle of starting from simple test items and increasing in complexity. This is achieved by ensuring that each test item contains only one distinguishing phenomenon (that is, only one phenomenon that distinguishes it from other similar test items). Progressivity necessitates that the design include a definitive list of hierarchically-arranged phenomena, and an analysis of each test item. The analysis must show each constituent, its instance and position in the sentence (what and where is it), its lexical or syntactic category (*e.g.*, noun phrase), and syntactic function (*e.g.*, subject) in the sentence.

Systematicity

Depth of coverage of well-formed and ill-formed test items depends upon their being systematically generated. In the TSNLP suite, systematicity is achieved for the well-formed test items by their explicit classification according to phenomena and sub-phenomena. In particular, the TSNLP data are designed to exhaustively test the treatment of closed class items in a language. General guidelines were determined to describe the closed class items that should be tested:

- Pronouns: personal, reflexive, possessive, relative, etc.
- Question Words: interrogative, pronouns, complementizers
- Determiners: articles, possessives, demonstratives, quantifiers, etc.
- Auxiliary verbs: copular verbs, modal verbs (including auxiliaries, passives, marginals)
- Conjunctions: coordination, subordination
- Adverbs: negation, common temporal and locative deictics[12]
- Prepositions

In the TSNLP suite the ill-formed items are systematically formed from the well-formed items through the use of four operations.

[12] A deictic is a word that points, indicates, or illustrates.

- Replacement (e.g., change the number of a subject noun phrase)
- Addition (e.g., add a noun phrase or object to a sentence with an intransitive verb)
- Deletion (e.g., delete an obligatory verb or noun phrase)
- Permutation (e.g., inverting the order of a verb and object)

These operations are applied in a precise and constrained manner to grammatical sentences, according to the phenomenon being tested. For example, agreement and complementation are common phenomena. In TSNLP the *agreement* phenomenon allows only the operation replacement of person or number, while *complementation* allows replacement, addition, and deletion. A grammatical sentence could possibly result from an operation, particularly the latter example, addition or deletion of an object in the case of verb *complementation* phenomena, in which case the sentence is not acceptable as an ill-formed item.

Systematicity of the test data was enhanced by construction and use of a test suite construction tool (tsct).

3.2.5 The TSNLP Annotation Scheme and Database Model

A detailed annotation scheme was designed for the TSNLP test data (Estival *et al.*, 1994b). The test data are *theory-neutral* (do not presuppose any particular linguistic theory of language). Estival *et al.* claim to have tried to be as neutral and uncontroversial as possible in choosing their list of syntactic categories and category names. Neither does the test data presuppose a particular evaluation type, purpose, user, or use. They do cite several authorities as sources of English grammar: Huddleson, Hornby, Svartvik, and Quirk. For most phenomena though, they follow Quirk *et al.* (1991), a standard academic grammar of English.

The test data and annotations are organized at several levels. These describe the entity sets of a relational database. An improper SQL subset language is provided so that the user may query the database. Relationship sets exist, but the user need not know about them. Their

SQL assumes a natural join if more than one entity is involved in a query. Here is a brief description of the structure of the database.

Input

The *input* table describes the core data, that is, the set of individual test items, sentences or phrases. The input table includes only the information about a test item that is independent of the phenomena that the item is designed to test. Phenomena-related data are contained in another table. The input table includes attributes such as a unique identification number; the author, date, and origin of the data; but the interesting attributes are:

- the sentence string
- length in words
- category: S for sentence, S_inter for questions, NP for noun phrase fragments
- a well-formedness code: 0 for ill-formed, 1 for well-formed

i-input	i-category	i-wf	i-length	i-comment
He comes .	S	1	2	
Him comes .	S	0	2	
The firm , that succeeds , comes .	S	1	5	Rare
Is he carrying it ?	S_inter	1	4	

Analysis

Each well-formed test item has an associated analysis that gives the syntactic category and string positions of its lexical and uncontroversial phrase elements. According to Lehmann *et al.* (1996b) the TSNLP group deliberately avoided using potentially controversial phrase structure representations, but instead provided a somewhat underspecified dependency or functor-argument structure that can be mapped into a constituent structure. The data fields in the *analysis* table are:

- The unique identifying number of the sentence (not shown here).

- The position, text, and category of the lexical and phrase elements.
- Function and domain annotations that describe the minimal dependency structure.

i-input	a-position	a-instance	a-category	a-function	a-domain
He and I succeed .	0:1	He	PRON_pers-3-nom-sg-m		
	1:2	and	COORD_conj		
	2:3	I	PRON_pers-1-nom-sg		
	0:3	He and I	NP	Subj	1:2
	3:4	succeed	V	Func	0:4
	4:4	.	PUNCT(.)		

Phenomenon

The phenomena set was selected on the basis of *linguistic relevance* and *frequency* for the individual languages. The resulting set includes: sentence and clause types; tense, aspect, and modality; diathesis[13]; complementation; modification; agreement; word order; co-ordination; negation; parentheticals; abbreviations.

The *domain* of a phenomenon is the syntactic category of the phrase in which it occurs (and is unrelated to the analysis table *a-domain*). Almost all of the phenomena can affect several categories, and the same syntactic category can be affected by more than one phenomenon. The list of TSNLP domains is as follows: S (sentence), C (clause), NP (noun phrase), AP (adjective phrase), ADVP (adverb phrase), and PP (prepositional phrase).

The TSNLP convention for naming phenomena is to prefix the name of the phenomenon with the name of the domain. Examples of the resulting phenomenon names are: *C_Complementation*, *NP_Coordination*, *NP_Agreement*. A more fine-grained division of phenomena is still required for some test items. For example, *C_Complementation* covers all of verb subcategorization. Estival *et al.* (1995) describe further subdivision of phenomena

[13] Diathesis is the phenomenon that changes the predicate structure of the verb. In English, the forms are active; passive; and transitive verbs having no agent and thus no passive form.

categories and phenomena naming conventions. An understanding of these conventions is required if the user wants to fully understand the phenomena associated with a particular test item.

The phenomena-related data are classified hierarchically. Each phenomenon described in the *phenomenon* table is identified with a name, unique phenomenon id, supertype, interaction with other phenomena, and any phenomena that must be presupposed. Phenomena are related to test items by a many-to-many relationship set.

i-input	p-name	p-presupposition
He comes .	C_Complementation-Monovalent-Subj(NP)	C_Agreement
He comes .	C_Diathesis-Active-Monovalent-Subj(NP)	C_Complementation, C_Agreement
He and I succeed .	NP_Coordination-Case_assignment-Subject_position	C_agreement, C_complementation
Is he carrying it ?	S_Types-Questions-Y/N_questions-Inverted	C_Agreement,C_Word-Order, C_Complementation

Ill-formed Test Data

A description of ill-formed test cases is provided in the *parameter* table. Similar to the analysis table for well-formed input, the data that are provided in the parameter table are the position and instance of the ill-formed phrase. A comment field identifies the exact nature of the error.

i-input	p-name	Position	Instance	Comment
Him comes.	C_Complementation-Monovalent-Subj(NP)	0:1	Him	wrong case

Test Sets

The test items may be grouped into *test sets*. A test set is a group of related test items containing both positive and negative test items for a common phenomenon.

User and Application Parameters

This contains information that relates a test suite to its evaluation purpose and application. Users of TSNLP who want to customize the suite to suit various purposes of their own may extend this part of the database themselves.

3.2.6 Construction of the TSNLP Test Data

Some five thousand test items were provided for each of the three languages. For English, there are 1582 grammatical and 3036 ungrammatical items. The data are described extensively in Lehmann *et al.* (1996a).

Vocabulary

The guidelines for test suite construction, which had been developed by TSNLP group, first required that a lexical domain be chosen. Then, the guidelines required that the vocabulary for the test items be selected from this domain. They decided to restrict the vocabulary to the "business" domain. Falkedal compiled an intersection of the vocabulary from five German dictionaries and translations were used to establish lists for English and French (Estival *et al.*, 1995). The translations were obtained from a dictionary that covers basic and general vocabulary. It was expected that the resulting word set would be reusable in each of the languages. As it turns out, only a fraction of the vocabulary was used.

Syntactic Categories and Functions

As stated above, the TSNLP scheme does not embody any particular linguistic theory, and the list of syntactic categories to be tested is as neutral as it can be. Estival *et al.* (1995) are careful to state that this list is not a proposal for a syntactic theory. It is a list of the labels needed to describe the test items. The major syntactic categories used in TSNLP are: sentence, clause, determiner, noun phrase, pronoun, adjective phrase, preposition phrase, adverb phrase, verb phrase, particle, complementizer, negation, coordination, comparative, and punctuation. The above category list gives only the category supersets. The actual TSNLP categories are numerous and descriptive (see Estival *et al.*, 1995).

Depth and interaction

The simplest test data were constructed one phenomenon at a time. The test item contained that phenomenon and as few others as possible. Complex constructions were avoided unless they were specifically being tested. If different variations of a phenomenon were possible, a test item was constructed for each variation.

Ill-formed input relevant to each individual phenomenon was derived.

Complex test data were created to allow for the co-occurrence of phenomena within a test item. Treating such interactions required the use of positive and negative "traps". A *negative trap* is an ungrammatical sentence that may be difficult to identify as such, for example, "*Either they or he are going*". A *positive trap* is a grammatical sentence that may mislead a system into identifying an error where there is none, for example, "*A great number of them are new*".

3.2.7 The Test Suite Construction Tool and the Test Suite Database

A graphical test suite construction tool (*tsct*) was implemented to facilitate test data construction, promote systematic production, and to reduce erratic variations in the instantiation of the annotation schema (Oepen *et al.*, 1995). The tool allows reuse of previously constructed and annotated data.

The TSNLP database was implemented as a relational database *tsdb* (Oepen *et al.*, 1995). Access to the *tsdb(1)* database is available through the Internet[14].

3.3 DiET – Diagnostic and Evaluation Tools for Natural Language Applications

The Diagnostic and Evaluation Tools for Natural Language Applications (DiET) project is the successor to TSNLP (Netter *et al.*, 1998). The main goal of DiET is to develop methods and tools for the glass box evaluation of NL components, and to provide flexible, user-friendly tools for the construction of test data. The annotation scheme and core test suites

[14] http://tsnlp.dfki.uni-sb.de/tsnlp/tsdb/tsdb.cgi?language=english

developed in TSNLP will be modified to provide a larger and more varied test set in the toolbox than supplied by TSNLP. Test data that have been systematically constructed are useful for diagnostic evaluation, but not necessarily for adequacy evaluation. The DiET project will address the issues involved in the customization of test material to specific domains and applications, lexical replacement tools, devices for document profiling, and methods for linking test suites to corpora.

4 A Test-Harness for DIPETT

One of the goals of this thesis was to produce an *improved* DIPETT. Delisle (1994) and Barker *et al.* (1998) showed that DIPETT delivers a parse analysis in a relatively short time, if it is going to at all. The goal of this thesis was thus to extend DIPETT's coverage, improve the quality of the parse trees generated, and to track successive revisions to ensure that each change caused no degradation.

Making such improvements to a broad-coverage natural language parser is a large task. The first step is to perform an evaluation, but that is a substantial goal in itself. This chapter describes the methods used for diagnostic and progress evaluation of DIPETT. It is appropriate to use a *test suite* (intrinsic, no-corpus evaluation; test suite based evaluation) for both diagnostic and progress evaluation, and possibly such performance measures as *coverage* and *average parse base* with respect to an unannotated corpus.

4.1 Review of existing test procedures, materials, and literature

The first phase in the evaluation and optimization of DIPETT was to select a test method and suite or suites that could be used for diagnostic evaluation. The first step was to examine the existing test materials that had been used during DIPETT's development, as well as the "real texts" on which DIPETT's performance had been tested.

4.1.1 Existing Test Suites

DIPETT's Input and Output

It is important to note that DIPETT's input is plain, unedited, untagged, text sentences. For each sentence, DIPETT produces a single parse tree based on its grammar and heuristics but no semantic information. When a full parse tree is generated, it is the *first* good tree that was found. This is not always a correct tree or the best possible tree. If a full parse tree cannot be generated, then DIPETT attempts to produce a fragmentary parse.

Test Sentences

Delisle (1994), the original author of DIPETT, developed a series of approximately 600 sentences that were intended to test every grammar rule in DIPETT with a specific test case, as in glass-box testing. Every rule is tested, but not their combinations. Delisle (1994) states that DIPETT produces a full parse tree for each one of those sentences, and that for 90% of the test cases the parse tree generated is the "best" tree.

Progress Testing

Delisle's test suite was used for progress evaluation, that is, to confirm during development that an extension to or change in the grammar did not affect the parse performance of the grammar as a whole. The only performance measure used for progress evaluation was *coverage*. That is, if 100% of the test sentences still produced full parse trees then a change to the grammar had not degraded its performance as a whole. No attempt was made to compare the parse trees generated by successive versions of the grammar to determine if the best or even the same tree was still being generated for each sentence.

4.1.2 Previous Test Studies

The QUIZ Manual and the Tax Guide

In his Ph.D. thesis, Delisle (1994) reports that extensive testing was performed on the QUIZ manual, a 135-page user's guide to Cognos' PowerHouse QUIZ report writer. The main text of the manual was processed, but not the appendices or any other non-textual material that was inappropriate to English parsing. Delisle also reports that parts of the 1990 Canadian Tax Guide were prepared and parsed. The purpose of these tests was to demonstrate coverage and performance.

Test-driving TANKA: clouds and small-engines

In 1996 Barker and Delisle and in 1998 Barker *et al.* published experimental results of TANKA's performance in extracting knowledge from two selected texts. In the first experiment, a simple child's science book describing a technical domain, the weather, was parsed by DIPETT. A syntactically more complex text was chosen for the second

experiment, a book on the mechanics of small engines. These experiments did not, and were not intended to, constitute a full-fledged diagnostic evaluation of DIPETT.

4.1.3 Use of the existing suites and studies for diagnostic evaluation

Generally, it was found that the existing test suite and text experiments were not useful for a full-fledged diagnostic evaluation. The general shortcomings found were:

- The suite of test sentences was designed to exercise DIPETT's rules, not to pinpoint grammar deficiencies.
- The test items seem biased toward the testing of syntactic constructions found in the QUIZ manual, rather than broad-coverage testing.
- No systematic testing of variations over a specific phenomenon.
- Lack of systematic testing of the co-occurrence of different constructions.
- Minimal documentation and annotation.
- The texts tested have all the problems of using a corpus as a diagnostic test tool: redundant coverage of phenomena, not broad-coverage, not systematic.

4.2 Considerations of Corpus-based vs. Test Suite based testing

The benefits and drawbacks of using a corpus were considered, and use of a corpus for testing of coverage was rejected for the following reasons:

- A corpus does not guarantee broad-coverage.
- A corpus lacks focus on particular phenomena.
- A corpus does not provide systematic testing of variations over a specific phenomenon.
- A corpus lacks systematic testing of the co-occurrence of different constructions.
- Typically, corpora are large.
- A corpus may provide redundant coverage of phenomena, in that the phenomena that do occur may occur repeatedly.

4.3 Proposal for Diagnostic and Progress Evaluation of DIPETT

4.3.1 Using the TSNLP Sentences

The TSNLP group has provided an excellent test suite for diagnostic evaluation. Certainly, a broad-coverage parser should be able to parse all of the TSNLP test sentences. However, the TSNLP test suite has limitations that arose from the design requirements:

- The lexicon is limited.
- The phenomena were chosen based on linguistic relevance and frequency *across three languages*, not just English.
- Interaction of phenomena is highly controlled.

In short, the TSNLP suite may be useful for highlighting deficiencies in a grammar, but it lacks the benefits of parsing corpus data and its coverage cannot be considered broad unless it were extended as envisaged by its design. Still, a diagnostic evaluation would be incomplete if the TSNLP data were not considered.

4.3.2 Proposal for a broad-coverage test suite

To augment the coverage provided by the TSNLP suite, I proposed to design a collection of test sentences that was annotated and organized according to a formal English grammar description. Quirk *et al.*'s grammar (1973, 1985) was chosen as the source. The plan was to extract the example sentences and the ill-formed[15] sentences from Quirk *et al.*, and organize them by topic. These would then be used to test DIPETT's performance on a distribution of syntactic structures.

The advantage of this method was that it drew upon the strengths of both test-suite methods and corpus-like methods:

- The test sentences are not derived from DIPETT's grammar.

[15] ill-formed sentences are conventionally annotated with a star.

- They provide broad coverage of linguistic phenomena and are indexed according to linguistic phenomena.
- They are rich in syntactic structure and test a distribution of syntactic structures.
- The grammar is buttressed by an authority.
- The test sentences use an unconstrained vocabulary.
- The coverage of phenomena *interaction* is corpus-like in that it is not controlled as in a test suite.

The disadvantages of using the Quirk sentences as a diagnostic tool are as follows:

- The lexicon is not limited and is therefore harder to manage.
- Sentences contain phrases that are irrelevant to the phenomenon described.
- Redundant structures and variable depth: some sections have many examples, some have very few.
- The complete collection is large.

4.3.2.1 Creating the Quirk test suite

After some consideration, it was decided to use the comprehensive Quirk *et al.* (1985) grammar, and not the abridged Quirk *et al.* (1973). The primary reason for that choice is that the abridged grammar is not properly broad-coverage. The abridged grammar does not cover marginal auxiliaries, for just one example.

Extracting test sentences from the larger comprehensive grammar was tedious, labour-intensive, and time-consuming. It was possible to extract the sentences from only three complete chapters, chapters two, three, and four. Even so, only the first two chapters extracted, chapters two ("A survey of English grammar") and three ("Verbs and auxiliaries"), are actually used in this thesis. Chapter four deals mainly with the semantics of verb phrases, and while the material covered is extremely interesting, it turned out to be not very useful in diagnosing a surface syntactic parser.

Both the Quirk test sentences and the TSNLP test sentences focus on the syntax of verb phrases and so, the relative merits of both can be compared.

The test suite extracted from Quirk *et al.* (1985) chapters two and three is shown in Appendix A. Most of the sentences are annotated with information regarding the phenomenon or phenomena that they illustrate.

4.3.3 DEVAL: A Program for DIPETT Progress Evaluation

DIPETT's progress evaluation testing scheme was minimal. It was beneficial to develop a software program that compares the performance of successive versions of DIPETT, to ensure that grammar changes do not have any unintended or adverse consequences. Thus, the DEVAL program was designed and implemented to perform progress evaluation for DIPETT.

4.3.3.1 Design Issues

Platform

DIPETT is implemented in Quintus Prolog, and local development is performed using Unix SunOS or Solaris on the KAML lab SparcStations[16]. The DIPETT parse tree output files are therefore Unix files of Prolog structures. There is no advantage to implementing a DIPETT evaluation program on another platform, but there would be disadvantages: the necessity of file transfers; lack of integration into DIPETT. Clearly, the DIPETT progress evaluation software should run on the DIPETT development machines.

Interface or Integrate?

DIPETT progress evaluations should be simple for the user to perform. It would be simplest for the user if progress evaluation were a one-step operation. DIPETT has two main modes of operation: a batch mode for processing an input file, and a user screen interface mode for parsing single sentences and performing other TANKA functions (such as semantic analysis). It is possible to integrate a self-evaluation module directly into DIPETT, and to

[16] Development is also performed at l'Université du Québec à Trois-Rivières.

have it accessible through the user interface. The disadvantage of such a complete integration would be increased DIPETT executable size and potential complications (user options set, semantic analysis turned off).

It would be possible to make evaluation of DIPETT progress a completely separate program, but this would necessitate interface through the parse tree output file and likely a two stage process (parse then compare). The disadvantage of this approach is that it is inconvenient for the user. DIPETT may take some hours to parse a batch of sentences and it would be preferable to start the evaluation process and have the report generated without user intervention.

A hybrid approach was chosen. The DIPETT evaluation module (DEVAL) was implemented as a standalone prolog program that contains its own user interface and also can be loaded into the DIPETT executable. Standalone, DEVAL can compare the parse trees in two separate files and generate a report of the similarities and differences. Loaded into DIPETT, DEVAL can direct DIPETT to parse a file of test sentences, and then perform the same comparison operation on the resulting parse tree file. The benefits of this approach are: a one-pass evaluation procedure, and no modifications required to DIPETT.

Method for Comparing Parse Trees

Various methods for comparing parse trees were considered. Is a two-valued same/different verdict sufficient? Should the complete parse tree be compared, or just the part of the parse tree that illustrates the phenomena being tested? It is simple to compare two structures using Prolog. Since the purpose of DEVAL is to report changes in parse tree output, a yes/no comparison procedure was chosen that provides an indication of where and how two parse trees differed if they were found to be different.

The algorithm for locating the first difference in a pair of dissimilar parse trees is a recursive depth-first search. Prolog data structures are converted to list structures so that the elements may be compared. The first difference found is reported.

```
IF at least one of the elements to be compared is atomic
OR at least one of the elements to be compared is an empty list THEN
   evaluate whether the elements are the same or not and
   return that value
ELSE IF the elements to be compared are both data structures THEN
     convert the data structures to list structures and
     compare those
ELSE {both are lists} IF the list head elements are not the same
THEN
        compare the head elements
     ELSE
        compare the list tails
```

4.3.3.2 DEVAL Output

If DEVAL is used with DIPETT to parse a suite of test sentences, then DEVAL saves the resulting parse trees and parse script files. DIPETT version information is saved in the parse script, as is a list of all the input and output file names and paths. The input files are the file of sentences parsed and the associated file of dictionary entries. The output files are the parse tree file, the parse script file, and DEVAL's comparison information.

DIPETT generates different parse tree types, depending on the sentence input and the success or failure of the parse. Examples of DIPETT parse tree types are:

- parse_tree__decla_or_imper — for declarative and imperative sentences.
- parse_tree__inter — for interrogative sentences.
- parse_tree__frag_series and parse_tree__frag — for fragmentary parses.
- error — for sentences that could not be parsed (usually results from a timeout).

DEVAL's comparison output provides the following information:

- The *file names* of the parse tree files that were loaded for the comparison and an indication if they were not loaded successfully.
- A *count* of the parse trees that were loaded from each of the files compared.
- An itemization of the number of *parse tree types* for both files.
- A list and count of sentences that were *not successfully parsed* in either file

- A list and count of sentences that were successfully *parsed in one file but not the other.*
- A list and count of sentences that were successfully *parsed in both files.*
- A *comparison* of the parse tree structures for the sentences that were successfully parsed in both files. First, a list is generated of sentences that were successfully parsed *and* the parse tree structures are identical in both files. Then, the successfully parsed but non-identical parse trees are compared. The sentence that was parsed is contained in the parse tree structure, and that is compared first. If the sentences are the same, then the parse trees are compared and the first difference found is reported.

4.3.3.3 DEVAL File Names

DEVAL expects all of the files for a test suite to be maintained together in a single directory, and each separate test suite have its own individual directory. DEVAL file names must adhere to the following specification:

File Type	*In or Out?*	*Name Specification*	*Example*
Sentences	Input	*dir*-sent.txt	ch2/ch2-sent.txt
Dictionary	Input	*dir*-dict.txt	ch2/ch2-dict.txt
Original Parse Trees	Input	*dir*-test-trees.pl	ch2/ch2-test-trees.pl
Output Parse Trees	Output	DIPETT default	ch2/zzz.DIPETT-HAIKU.OutStructs.pl
Comparison Script	Output	DIPETT default	ch2/zzz.DIPETT-HAIKU.script

5 A Diagnostic Evaluation of DIPETT

DIPETT v3.0 was evaluated using sentences extracted from Quirk *et al.* (1985) and sentences extracted from the TSNLP database. This chapter presents DIPETT's performance on both of the test suites and a discussion of the DIPETT problems identified.

5.1 Evaluation Method

5.1.1 Sentence Extraction

Quirk Sentences

Sentences were extracted from Quirk *et al.* (1985) chapters 2 and 3 as described above in section 4.3.2.1 and as shown in Appendix A. The sentences were augmented with selections from chapter 5 of Quirk *et al.* (1985) that illustrate article usage. Both grammatical and ungrammatical example sentences were included.

TSNLP Sentences

The complete *tsdb(1)* data were downloaded and stored as a Microsoft Access database. Queries were developed to extract test items and their related phenomena and analysis. The TSNLP database is large — 4612 sentences and sentence fragments in all. The data used for this thesis was restricted to the set of test items that are complete sentences and clearly either grammatical or ungrammatical. That is, items having:

- i-category: *S* or *S_inter*
- well-formedness code: zero or one

The extraction produced 1173 grammatical and 1535 ungrammatical sentences.

5.1.2 Sentence and Lexicon Preparation

DIPETT takes text files of sentences as its input. Each sentence must be terminated by a period, ellipsis, question mark, or exclamation mark. The text file must be terminated by a particular end of file marker.

A lexicon must be produced for each batch file of sentences. DIPETT uses the Collins wordlist to create an auxiliary dictionary file that contains all possible part of speech entries for all of the unknown words in the input sentences.

A small Windows program, DICTMAN, was written to aid in the preparation of DIPETT input files, to ensure that the sentence batch files and auxiliary dictionary files were properly formed. Appendix G shows the DICTMAN user interface. The sentence and dictionary preparation program performed the following functions for the sentence batch files:

- Checked that a valid terminator terminated each sentence.
- Marked the lines that contained more than one terminator.
- Flagged any duplicate sentences.
- Flagged the starred sentences.
- Appended the DIPETT's end of file marker to the batch file.

DIPETT's code includes a built-in dictionary of over 3000 dictionary entries (see section 6.3). The internal dictionary includes many closed class items, words commonly used, and idioms. One of DIPETT's known problems is that the built-in dictionary does not necessarily contain all of the potential parts of speech for open class items. The DIPETT code that creates auxiliary dictionary files from Collins does not extract entries for words that are already known to DIPETT. A sentence may contain a word that is known by DIPETT, but not as the part of speech used in that sentence. In that case, the sentence will always fail to parse unless the auxiliary dictionary is augmented.

DIPETT's idiom entries are common collocations with clear syntactic function. Idioms are specially coded in DIPETT's built-in dictionary and must be coded in the same fashion in the input batch files of sentences.

If the DIPETT code that creates auxiliary dictionary entries from Collins cannot find an input word, it assumes that it is a *propernoun*. That assumption is not always correct.

The sentence and dictionary preparation program DICTMAN performs the following functions for the auxiliary dictionary files:

- Relates the dictionary entries for any word with all the sentences containing the word.
- Allows editing and insertion of dictionary entries.
- Identifies words that have multiple part-of-speech entries, words entered as both a noun and a verb, and words that are entered only as a proper noun.
- Identifies sentences that contain an idiom that is coded into DIPETT's built in dictionary.
- Allows editing of sentences to conform to DIPETT's idiom conventions.

5.2 Summary of Parse Results

Sentence batch files and auxiliary dictionary files were created for the Quirk sentences and TSNLP sentences, and DIPETT[17] batch jobs were run to parse the sentences. Appendix B shows the details of DIPETT's parse results on the Quirk sentences. Appendix C shows DIPETT's parse results on the full TSNLP phenomena. Appendix C1 shows the details of DIPETT's performance on the TSNLP tense-aspect-modality combinations.

The parse results for the grammatical sentences are summarized in Table 5-1 and the parse results for ungrammatical sentences are summarized in Table 5-2.

Grammatical Sentences

Coverage is the percentage of examples from a given corpus for which a parser/grammar is able to assign at least one complete (not fragmentary) analysis. DIPETT produced full parse trees for 79% of the grammatical Quirk sentences but only 65% of the grammatical TSNLP sentences. This result was unexpected because the TSNLP sentences are shorter, simpler, and narrower in their coverage of phenomena and phenomena interaction than are the Quirk sentences.

[17] "DIPETT" referred to in this thesis chapter is DIPETT v3.0.

Ungrammatical Sentences

Ungrammatical sentences should be rejected by a parser/grammar. DIPETT failed to parse (generated a fragmentary parse for) 38% of the Quirk sentences and 53% of the TSNLP sentences. It was found that many of the ungrammatical sentences were not useful for diagnostic evaluation of DIPETT; ungrammatical sentences are useful only for testing phenomena that are specifically rejected by the grammar. For DIPETT, ungrammatical sentences are useful for testing such phenomena as number agreement but not phenomena such as complementation. Sometimes the full parse trees generated for ungrammatical sentences are perfectly reasonable, given the information DIPETT has available. This is particularly true for the Quirk sentences, where some of the ungrammaticality is purely semantic, for example:

* The bus hopes to be here at five.

Table 5-1: DIPETT v3.0 — Parse Results for Grammatical Sentences

Grammatical Sentences	Quirk *et al.* chapter			TSNLP
	2	3	5	
Number of grammatical sentences	252	308	18	1143
Number of full parses generated by DIPETT	218 (86%)	221 (72%)	18 (100%)	745 (65%)
Number of correct full parses	120 (48%)	117 (38%)	18 (100%)	467 (41%)

Table 5-2: DIPETT v3.0 — Parse Results for Ungrammatical Sentences

Ungrammatical Sentences	Quirk *et al.* chapter			TSNLP
	2	3	5	
Number of ungrammatical sentences	22	44	10	1519
Number of fragmentary parses	7 (32%)	17 (39%)	5 (50%)	800 (53%)
Full parses of ungrammatical sentences	15 (68%)	27 (61%)	5 (50%)	719 (47%)

5.3 Evaluation of DIPETT's Performance on the Quirk Sentences

5.3.1 Summary of Quirk Sentences Results

Appendix B shows the details of DIPETT's parse results on the Quirk sentences. The results shown for each sentence are a representation of DIPETT's parse tree output, comments on parse tree errors, and an indication of whether the parse tree is correct. The problems identified in Appendix B are discussed below. Specific grammar rule problems are identified where appropriate and possible solutions are suggested. Some of the problems can easily be fixed but others are more difficult, particularly those that occur because DIPETT has very little specific word knowledge available. Obviously there are limits to the performance that can be achieved with a surface syntactic parser, more so a parser that outputs only the first parse tree found.

5.3.2 Evaluation on Quirk Sentences: Problems Identified

5.3.2.1 Adjectives overgenerate from adverbs

DIPETT's noun phrase grammar contains a rule that states that adjectives can be formed from adverbs.

```
adj_equiv(Adj, pos)  -->  adv_s(Adj)
```

This rule is likely responsible for overgeneration of adverbs from adjectives. For example, in the sentence *"The evenings have turned very cold just recently."* DIPETT's parse tree has the adverb phrase *"just recently"* interpreted to be an adjective phrase, conjoined with *"very cold"*.

An adverbial interpretation for adverbs should be attempted before their conversion to an adjective is considered.

5.3.2.2 Intensifiers of adjectives and adverbs are conjoined with the head

DIPETT does not distinguish dependents from their heads in adjective and adverb phrases. For example, in the sentence *"The evenings have turned very cold."* the adjective phrase is

parsed as consisting of adjectives *very* and *cold*, assumed joined by an implicit *and*. This interpretation is incorrect, since *very* is an intensifier of *cold* and cannot stand alone.

DIPETT generates successful parses for all three of the following sentences, although only the first is properly grammatical.

The evenings have turned cold.

* The evenings have turned very.
* The evenings have turned very and cold.

DIPETT's internal dictionary distinguishes categories such as predeterminers, determiners, cardinal numbers, ordinal numbers, and quantifiers. Intensifiers could be similarly classified, rather than simply entered into the general-purpose adjective category as a positive adjective.

A related problem also arises, that is, intensifiers may also be mistaken to be noun postmodifiers. For the sentence *"My mother enjoys parties very much"* DIPETT mistakes the intensifier *very* to be a postmodifier of the head noun *parties*. DIPETT's rule that allows this to occur is perfectly acceptable:

```
noun_post_modifier(adj(ADJ,DEG))        --> adj(ADJ,DEG).
```

Classification of *very* as an intensifier in preference to a general adverb would allow its correct attachment here as well.

5.3.2.3 Adverb attachment to prepositional phrases

Adverbs followed by a prepositional phrase are displayed in the parse tree as if they are interpreted as the preposition, and not attached to the main verb. For example, DIPETT's parse tree for the sentence *"Someone was laughing loudly in the next room."* shows the prepositional phrase *"in the next room"* as being marked by the adverb *loudly* and not the preposition *in*, although the tokens shown for the phrase are *"loudly in"*. The preposition *in* does not appear elsewhere in the parse tree.

```
[...]
pp  loudly
  tokens  loudly  in
  [...]
```

This problem occurred in three sentences of the Quirk test suite, and in each of those it would be appropriate if the adverb were interpreted to modify the main verb and not the preposition that follows. Quirk *et al.* (1973; 5.25) states that few adverbs premodify particles in phrasal verbs but those that do also premodify prepositions or prepositional phrases. Quirk *et al.* (1985) does not offer more than that. From this it can be concluded that adverbs that precede a preposition should be attached to the main verb, and not the preposition. Some experimentation would be necessary to verify this.

DIPETT does not parse the adverb before it looks for the prepositional phrase complement, so the adverb is necessarily parsed by the prepositional phrase rule that allows a required adverb before the preposition:

```
prepphrase_core(pp([Advs,P],Np),Restriction) -->
          required_adverbs(Advs),
          prep_pp(P,Restriction), np_for_pp(Restriction,Np).
```

There is no documentation in DIPETT to cross reference this rule to a specific section in Quirk *et. al.* nor is an example sentence given. It seems likely, though, that this rule is intended to parse only the small subset of adverbs which do modify prepositions, for example, *"It drove me right up the wall."* If this is indeed the case, then it seems that those adverbs could be classified in DIPETT's dictionary, in the same manner as are prepositions that function as adverb particles.

Further, important syntactic markers such as prepositions should not be "lost" in the parse tree representation. The parse tree output routines do not adequately handle the above prepositional phrase output structure `pp([Advs,P],Np)`.

5.3.2.4 Prepositional phrases are not recognized as adverbs

DIPETT's verb complement grammar is nonspecific with respect to objects and adverbials, generating **SVOO_SVOC_SVOA** for ditransitive verbs and **SVC_SVA** structures for copular verbs.

That strategy appears to be quite effective for ditransitive and copular verbs. However, the sentence *"Her father works in a factory."* is incorrectly assigned an **SVO** structure. In such cases, prepositional phrases are almost always incorrectly interpreted to be an object rather than an adverbial. DIPETT's parse for the above example sentence shows clause structure:

S her father **V** works **O** *PP* in a factory

Quirk *et al.* (1973; 2.11 and 1985; 2.24) states that an object complement is normally a noun phrase or a nominal clause. There is no indication that a prepositional phrase can serve as an object unless it is the indirect object of a ditransitive verb and appears immediately after the direct object. Quirk *et al.* also states that an adverbial is normally an adverb phrase, a prepositional phrase, or an adverbial clause, or possibly a noun phrase.

Contrary to Quirk *et al.*'s definition, DIPETT's implementation of **SVO** structures allows the object to be either a noun phrase or a prepositional phrase. This definition gives rise to incorrect interpretation of prepositional phrases as objects.

```
complement_tr(active, nil , svo(X)) -->
   (np_equiv(post_v, 0, X ,_) ; required_prepphrases(X, _))
```

DIPETT v3.0 contains no rule that would generate an intransitive **SV{A}** structure from a prepositional phrase complement.

5.3.2.5 Time-related and place-related noun phrases are not recognized as adverbs

Time and place related noun phrases often function as adverbs in a sentence, but are not recognized as such by DIPETT. Noun phrases such as *"every year"* and *"this afternoon"* are parsed as noun phrases, which in DIPETT's grammar do not function as adverbials. DIPETT's grammar is unable to completely analyze sentences such as *"They make him the chairman every year."* which should, according to Quirk *et al.* (1985; 2.45) be parsed as an **SVOC{A}** structure. DIPETT can, however, parse *"They make him the chairman annually."*

Quirk *et al.* (1973; 2.11 and 1985; 2.24) states that an adverbial can possibly be a noun phrase. DIPETT's adverb grammar is convoluted and difficult to follow; there is no indication that DIPETT does indeed allow a noun phrase to be interpreted as an adverb.

Since noun phrases are never interpreted to be adverbials in DIPETT, they can also give rise to incorrect interpretation of actual noun phrases. For example, in the sentence *"We returned from Italy last week."* both of the words *Italy* and *last* are analyzed as premodifiers of the head noun *week*.

According to Quirk *et al.* (1985; 9.40), it is possible to identify time-related noun phrases. Time-related noun phrases are marked by the absence of a preposition immediately before a deictic word such as *last*, *next*, *this*, or *that* or before the quantitative words *some* and *every*. A syntactic solution is therefore possible for time-related phrases such as *"last week"* and *"every year"*.

5.3.2.6 Adverbs may be misinterpreted to be an intensifier

Not only can intensifiers be mistaken as adverbs as discussed in 5.3.2.2, but adverbs may be misparsed as intensifiers. DIPETT parses the sentence *"Usually, my mother enjoys parties."* correctly. If the comma is removed, however, then *usually* is parsed as an intensifier of the entity *"my mother"*.

```
subj
  entity
    intensifier  usually
    determinatives
      deter  my
    head_noun
      noun
        n  mother  countnoun
        number  sg3
  tokens  usually  my  mother
```

This shows that DIPETT allows an intensifying adverb to appear before the determinatives in a noun phrase nucleus. Quirk *et al.* (1985; 7.62,7.63,7.68,17.111) does describe premodification of noun phrases by adverbs, but only as a "minor type" of premodification,

and also states that very few adverbs premodify nouns and precede the determiner in doing so. Some examples are: "*quite a party*", "*such a good boy*", "*rather a mess*". Intensifying adverbs may premodify pronouns and predeterminers: "*nearly everybody*", "*roughly half the equipment*", "*over two hundred dollars*".

There is no specific cross-reference or example offered by DIPETT to explain its rule that allows any adverb to appear before the determinatives of a noun phrase and act as an intensifier. The rule does appear to be overly permissive:

```
np_nucleus(1,entity(intensifier(Adv),Dets,ATTRS,HEAD,number(N)),N) -
->
  ({Adv = nil} ; adv_s(Adv)),
  determinatives(Dets, Num1),
  {adv_det_filter(Adv, Dets), predeter_filter(Dets)},
  rest_common_noun_np(Dets, Num1, ATTRS, HEAD, NUM), {!}.
```

Most likely, DIPETT should parse phrases like "*Usually my mother*" with the adverb separate from the noun phrase.

5.3.2.7 Predeterminer position is restricted

Quirk *et al.*'s (1985; 5.16) noun phrase grammar specifies that the predeterminers *all* and *both* can occur after the head of the noun phrase and within the predication in the medial adverb position.

DIPETT's grammar allows only the first of these examples:

- ✔ Both the students passed the exam.
- ✘ The students both passed the exam.
- ✘ The students can both pass the exam.

Extending DIPETT's noun phrase grammar to parse the latter two examples should present no problem.

5.3.2.8 Object complements not recognized

DIPETT outputs the first full parse tree that it finds, and it looks for verb sequences in the active voice before looking for stative verb sequences. Superficially, the following sentences have exactly the same structure:

subject	*verb*	*object*	*object*
↓	↓	↓	↓
They	consider	him	a genius.
They	threw	him	a lifeline.

The verb *consider* is stative, however, and the first sentence has an $\mathbf{SVO}_{\text{direct}}\mathbf{C}_{\text{object}}$ structure[18] (Quirk *et al.* 1985; 2.17). The verb *throw* is active and the second sentence has an $\mathbf{SVO}_{\text{indirect}}\mathbf{O}_{\text{direct}}$ structure. DIPETT does not make this distinction. If both verbs are entered into the dictionary as transitive then DIPETT generates the same active **SVOO** analysis for both sentences. DIPETT fails to parse *"They consider him a genius"* if the verb *consider* is entered into its dictionary only as a stative verb.

This situation can be improved upon for verbs that are known by DIPETT to be both transitive and stative. The dual classification carries useful information: such verbs are likely to be complex transitive and not ditransitive, or else they should not be classed as stative. In this case, DIPETT should reject the initial $\mathbf{SV}_{\text{active}}\mathbf{O}_{\text{indirect}}\mathbf{O}_{\text{direct}}$ analysis in favour of the more favourable $\mathbf{SV}_{\text{stative}}\mathbf{O}_{\text{direct}}\mathbf{C}_{\text{object}}$ interpretation that will subsequently be generated.

If a sentence has only a single noun phrase complementing the verb *consider*, then that noun phrase is a direct object and not a subject complement, for example, *"They are considering him"*. It may be possible to generalize that transitive-stative verbs are also monotransitive and not copular, but some experimentation is required to verify this.

The same problems exist for passive sentences. DIPETT's analysis of the sentence *"He was considered a genius."* identifies the subject complement *He* to be an object entity in an

[18] O_{direct} is a direct object. $O_{indirect}$ is an indirect object. C_{object} is an object complement.

$S_{nil}V_{passive}OO$ structure, and the two objects are not further differentiated. The above solution for the active voice also applies to the passive.

5.3.2.9 Date form not recognized

DIPETT's grammar recognizes a few date constructions. It does not recognize phrases in the form *"the fifth of February"*. The phrase actually contained in the example sentence is *"the 5th of February"*. DIPETT's scanner does not recognize ordinals in the form *5th*.

5.3.2.10 Phrasal quantifiers not recognized

Quirk *et. al.* (1973; 4.14) describe a large open class of phrasal quantifiers. A few examples are:

plenty of furniture
lots of students
a great deal of money

DIPETT's grammar allows some partitive structures but not these phrasal quantifiers. Their regular structure, described in detail by Quirk *et al.*, suggests that some syntactic solution may be possible. When DIPETT misses a partitive structure, it generates a noun phrase with a prepositional phrase postmodifier. The number of the noun phrase depends on the head noun, but is most often incorrectly interpreted as being singular, as in the following erroneous analysis of the phrase *"a great number of them"*:

DET a *ADJ* great *HEAD-NOUN-sg* number *POSTMOD* [*PP* of them]

If such a noun phrase is in a constituent position that is sensitive to number agreement then DIPETT fails to parse the sentence.

5.3.2.11 Count nouns and zero article

DIPETT's dictionary recognizes countnouns, massnouns, and propernouns. Superficially it appears that DIPETT's grammar is permissive in its treatment of countnouns and determiners; DIPETT accepts ungrammatical noun phrases such as *"I saw book"* (Quirk *et al.* 1985; 5.1). DIPETT's grammar can easily be made to reject such constructions. That sort of

modification may be completely undesirable, however, unless some mechanism is also constructed to distinguish nouns and/or syntactic structures that can properly take zero article. Identifying nouns that admit zero article is an open problem; article contrast is a semantic issue, not syntactic.

5.3.2.12 Collective nouns not recognized as being plural

DIPETT has no mechanism for identifying collective nouns as plural unless they have been entered into the dictionary as either a plural countnoun or as a massnoun. Neither strategy is correct. DIPETT expects plural countnouns to be associated with a singular form, which in the case of collective nouns does not exist. Massnouns are treated as a singular undifferentiated body, not as a plural.

The sentence *"The police want him."* provides an example. Quirk *et al.* (1985; 5.78) classify the word *police* used in this sense as a collective noun. If *police* is entered into the dictionary as a massnoun then the sentence fails to parse, as it should. If *police* is entered as a plural countnoun that is the plural of *police* then a full parse tree is generated but the parse tree suggests the existence of a singular form ("a police", for example) . If *police* is entered as a plural with an unspecified singular form, then the parse tree fails to print the entity head noun properly — the root of the head noun is shown as:

```
n  UNBOUND  countnoun
```

Similar problems have recently been reported with summation plurals and other 'pluralia tantum'[19] (Quirk *et al.*, 1985; 5.76-77).

5.3.2.13 Relative pronoun not recognized

DIPETT's grammar recognizes noun phrase postmodifiers marked by relative pronouns. The list of relative pronoun markers is not complete, for example, the word *where* is missing and there may be others. DIPETT is unable to parse the sentence *"The place where Mary lives is in London."*

[19] Ken Barker recently reported problems with the phrase "thanks in advance".

This can be fixed by adding *where* to the list of relative pronouns in DIPETT's dictionary, but some consideration must be given to whether or not that single addition makes the list properly complete. Quirk *et al.* (1985; 6.33) classify restrictive and nonrestrictive relative pronouns (*who*, *whom*, *whose*, *which*, *that*, and "zero"). Quirk *et al.* (1973; 13.7) explain that relative pronouns can be replaced by special adjunct forms for place, time, and cause (*where*, *when*, *why*).

5.3.2.14 Postmodifiers and complements

DIPETT's noun phrase grammar contains simple heuristics to help identify whether phrases should be identified as a noun postmodifier or as a complement. The heuristics are not always completely accurate. In particular, noun phrases that appear after the main verb can have a prepositional phrase postmodifier misattached as a verb complement. For example, prepositional phrases marked by *in* or *with* are assumed to be verb complements, not noun phrase postmodifiers.

DIPETT analyzes the sentences "*I remember that girl with the red hair*" and "*I remember all those fine warm days in the country last year*" as **SVOO_ SVOC_ SVOA** structures:

S ! V remember O that girl O/C/A with the red hair

S ! V remember O all those fine warm days O/C/A in the country last year

In both cases the prepositional phrases should be a postmodifier and not a separate object or complement (Quirk *et al.*, 1985; 2.28):

S ! V remember O that girl *POSTMOD* with the red hair

S ! V remember O all those fine warm days *POSTMOD* in the country last year

Conversely, complements may also be misanalyzed as postmodifiers. DIPETT analyzes the following sentence with a *that-clause* postmodifier of *trip* but the *that-clause* should be an adjective complement of *best* according to Quirk *et al.* (1985; 2.28).

DIPETT	S I V remember O the best trip *POSTMOD* that I ever had
Quirk *et al.*	S I V remember O the best trip C that I ever had

It should be noted that DIPETT was designed to be used with a post-editing module to correct misattachment problems, and places prepositionals at the top level for consistency (even if that is not always correct). The reattachment module was never built, however, and it is likely not possible that this problem can be completely solved without semantic knowledge or human intervention. Still, it may be possible to improve the heuristic functions.

5.3.2.15 Obligatory adverbials not recognized; obligatory ditransitive verbs not recognized

DIPETT does not recognize obligatory adverbials. Parse trees for sentences such as *"You must put the toys upstairs immediately"* (Quirk *et al.* 1985; 2.18) do not distinguish between the obligatory adverbial *upstairs* and the (optional) subordinate adverbial *immediately*. Similarly, for the sentence *"You must put the toys upstairs."* an **SVO** structure with a final subordinate adverbial is generated rather than an **SVOA** structure. The ungrammatical sentence *"You must put the toys."* parses successfully with an **SVO** structure.

Similarly, DIPETT recognizes verbs that are transitive, intransitive, and stative, but does not recognize the verbs that must be ditransitive. DIPETT parses correctly the sentence *"He allowed me a respite."* with an **SVOO** structure. It also parses successfully the ungrammatical sentence *"He allowed me."*, by incorrectly assigning an **SVO** structure.

As DIPETT is a shallow surface parser, these particular problems may be impossible to fix. One solution would be to extend DIPETT's simple verb dictionary with a full subcategorization dictionary. Another solution could be to have DIPETT learn verb subcategorizations much as HAIKU (the TANKA semantic interpreter) learns semantic case patterns.

Perhaps one solution is to ignore DIPETT's performance on ungrammatical sentences. DIPETT is intended to be a parser, not a grammar checker.

5.3.2.16 Conjunction misattachment

DIPETT is designed to handle conjoined structures. Almost any kind of structure can be joined by a coordinating conjunction, that is, full clauses, verb phrases, noun phrases, prepositional phrases, adjectives, adverbs. Delisle (1994) describes a fairly sophisticated heuristic mechanism for identifying compound sentences, but DIPETT does not always identify correctly which phrases or clauses should be joined by a coordinating conjunction.

These heuristics involve counting the number of potential verbs and coordinators in a sentence, and calculating the verbal density (roughly defined as the verb distribution in the sentence). The heuristics seem to work well for verbs in the active and passive voice, but the verb counting function does not appear to count stative verbs properly.

The verb counting function *evaluate_nb_potential_verbs* returns the following verb counts for some extremely simple test sentences.

Sentence	*Voice*	*Verb count*	*Parse tree correct?*
1. They went there and they bought it.	active-active	2	yes
2. It was visited and it was bought.	passive-passive	2	yes
3. It was red and they bought it.	stative-active	1	unable to parse
4. They bought it and it was red.	active-stative	1	unable to parse
5. It was red.	stative	0	yes
6. They bought it.	active	1	yes

Compound sentences are not parsed correctly if any of the clauses are stative.

5.3.2.17 Can't parse "be being"

DIPETT is unable to parse verb phrases in the form *be* plus *being*, for example:

> The ship may have *been being* sunk.
> He *is being* naughty again.

The cause of this problem is that DIPETT's dictionary does not contain *being* as a progressive stative form of *be*.

5.3.2.18 Complex transitives are parsed as nominals

Quirk *et al.* (1985; 15.3) describe nominal clauses as falling into six major categories, including *to*-infinitive and *-ing* clauses. They show that nominal clauses may function as subject, object and complement. Yet, their treatment of complex transitives (Quirk *et al.*, 1985; 16.49-54) makes no mention of the nominal. Complex transitive complementation occurs when the direct object is followed by a nonfinite clause that is also part of the predication. The object noun phrase is typically the subject of the nonfinite clause, commonly called a *raised* object.

DIPETT treats all nonfinite clause complements as though they were nominals, and does not provide for complex transitives as described by Quirk *et al.* Raising of the object is not shown in the parse tree. Quirk *et al.* (1985; 16.49) specifically list verbs and verb categories that may take various forms of nonfinite clause complements. For example, one variant of complex transitive complementation is *object+bare-infinitive*:

Verb Type	*Verb Examples*	*Example Sentence*
Verbs of coercive meaning	have, let, make	S Don't V let O him C sit like that
Perceptual verbs	feel, hear, notice, observe	S I V saw O her C leave
Verbs that also may take a *to*-infinitive	help, know	S He V helped O me C (to) edit it

This sort of information could be coded relatively easily in DIPETT by implementing the complement classes in the grammar and adding the verb classes to DIPETT's dictionary.

5.3.2.19 Passives are misinterpreted as being stative

DIPETT very often does not recognize past passive verb sequences. Usually, sentences such as "*the accident was seen*" are parsed with *was* being the stative main verb, and the past participle *seen* being its adjectival complement.

5.3.2.20 Complex to-infinitives not parsed

Quirk *et al.* (1985; 3.56) show how the aspect and voice auxiliaries *have* and *be* can occur in complex nonfinite verb phrases. The perfective, progressive, and passive forms should be allowed singly and in combination.

DIPETT's grammar allows only simple infinitives:

```
to_infinitive_clause(Type, to_infinitive_clause(X))                    -->
  simple_to_infinitive_clause(Type, X), {!}.

simple_to_infinitive_clause(Type, X)                                   -->
  infinitive_marker, verb_clause(Type, _, _, [inf], nil, _, X, nil).
```

DIPETT parses sentences such as *"I'm glad to have been working."* with "*been working*" as the object of the infinitive *have*.

Proper parsing of complex *to*-infinitives requires that DIPETT's *to*-infinitive clause rules be extended to allow complex infinitives, with the tense parameter of *verb_clause* not restricted to `[inf]`. DIPETT's verb sequence would also need to be extended, but this presents no particular problem. The verb sequence grammar for complex infinitives is essentially the same as that for verb sequences beginning with an auxiliary modal, for example:

Tense	*With modal auxiliary*	*In the complex infinitive*
Perfective	He *can* **have succeeded.**	His goal is *to* **have succeeded.**
Progressive	He *can* **be succeeding.**	His goal is *to* **be succeeding.**
Perfective+progressive	He *can* **have been succeeding.**	His goal is *to* **have been succeeding.**
Passive	He *can* **be examined.**	His goal is *to* **be examined.**
Perfective+passive	He *can* **have been examined.**	His goal is *to* **have been examined.**
Progressive+passive	He *can* **be being examined.**	His goal is *to* **be being examined.**
Perfective+ progressive+passive	He *can* **have been being examined.**	His goal is *to* **have been being examined.**

5.3.2.21 "used to" is not recognized in some instances

DIPETT's verb sequence grammar recognizes the marginal modal "*used to*", for example, "*She used to attend regularly*" (Quirk 1985; 3.44). A problem arises for sentences containing "*used to*" in the passive or stative because DIPETT's verb phrase grammar tries to match verb sequences first in the active voice, and then in the passive, and in the stative only if the previous attempts fail. For passive and stative sentences involving "*use to*" or "*used to*", DIPETT usually finds and accepts an incorrect active interpretation with the main verb *use*

and with *to* as a to-*infinitive* marker or a preposition before a passive or stative analysis is attempted.

This problem never occurs in the active voice because marginal modals are always parsed before the main verb. In that case "*used to*" is parsed first as a marginal modal, and only if that fails is *use* accepted as the main verb of the sequence.

The problem can be remedied by checking for passive and stative interpretations of "*used to*" before searching for an active interpretation.

5.3.2.22 Does not recognize "get" as a passive auxiliary

Quirk *et al.* (1985; 3.66) describe some circumstances under which *get* performs the function of a passive auxiliary "*The cat got run over (by a bus)*". DIPETT's dictionary does not contain *get* either as a passive auxiliary, or as a stative. A solution for this problem is described in section 6.3.7.

5.3.2.23 Multiple adverbs might be written incorrectly in the parse tree

DIPETT does not write multiple adverbs that are not immediately consecutive within the predication properly.

For the sentence "*I would probably never have believed that story*" DIPETT collects the adverbs into a list structure showing conjoined adverbs:

```
[conj_advs(implicit_and,[probably, never]), tokens([probably,
never])]
```

For this structure, the parse tree correctly displays:

```
advs
  conj_advs  implicit_and  probably  never
  tokens  probably  never
```

For the sentence "*I would probably have never believed that story*" DIPETT collects the adverbs into a list structure showing multiple but not conjoined adverbs:

```
advs([probably, tokens([probably]), never, tokens([never])])
```

The parse tree for this adverb structure does not properly represent the appended adverb. The second adverb *never* appears as though it is one of the tokens for the adverb *probably*. It should appear separately of its own accord, or listed as one of the *adv*, but certainly not as one of the tokens for *probably*.

```
advs  probably
  tokens  probably  never
  tokens  never
```

This problem arises because the parse tree printing routines are unable to handle the adverbial's list structure. This is therefore not considered a serious error, as the data structure generated by the grammar is correct. The problem can be fixed by modifying the parse tree printing routines *prt_node* and *prt_args*, or by changing how adverbs are appended so that the parse tree printing routines can properly display them.

5.3.2.24 Does not recognize some modal idioms, semi-auxiliaries, and catenatives

DIPETT's basic verb sequence grammar is based on Winograd's (1983) description. DIPETT documentation specifies that the grammar allows modal idioms "*would rather*" and "*had better*", and the semi-auxiliary "*be going to*".

Quirk *et al.* (1985; 3.45-49) list many modal idiom, semi-auxiliary, and catenative constructions. DIPETT performed poorly on the example sentences intended to test such structures (see Appendix B). For example:

- DIPETT does not recognize the modal idiom "*have got to*".
- The modal idiom "*would rather*" is not recognized if *would* is contracted.
- DIPETT does not recognize the semi-auxiliaries "*be bound to*" or "*be able to*" or "*be allowed to*".
- DIPETT does not recognize the semi-auxiliary "*be going to*" when it occurs in passive verb sequences or an interrogative sentence.
- DIPETT does not recognize catenatives "*appeared to*", "*failed to*", "*come to*", and others listed by Quirk *et. al.*

If DIPETT is to be considered a true broad-coverage parser of English then the constructions listed by Quirk *et al.* must be added to the verb sequence grammar. Adopting Delisle's method of implementing modal idioms and semi-auxiliaries directly into the code would likely result in an unwieldy and unreadable grammar. As far as possible, these constructions should be generalized into grammar rules that operate on dictionary entries.

5.3.2.25 Other linguistic phenomena not covered by DIPETT's grammar

The following are covered by the Quirk sentence test suite but are not covered by DIPETT's grammar. All section numbers are from Quirk *et al.* (1985):

- Pro-forms (2.44)
- Fronting (2.59, 18.20)
- Subjunctive in any form (3.59-62)
- Prepositional verbs (3.69)
- The passive gradient and verbs that cannot be passivized (3.68, 3.74)
- Operator in reduced clauses (3.26)
- Directives as distinguished from exclamatives (2.57)
- Proper names may consist of given name(s) and surname together (5.66). DIPETT parses the surname as an appositive of the given name, for example, the full name "Mary Smith" is parsed — *proper_noun* Mary *postmodif appositive* Smith

5.4 Evaluation of DIPETT's Performance on the TSNLP Sentences

5.4.1 Summary of TSNLP Sentences Results

Appendix C shows a complete summary of DIPETT's parse results on TSNLP grammatical sentences grouped by TSNLP phenomena. Table 5-3 shows a brief summary of Appendix C. Appendix C1 shows specific details of DIPETT's performance on the tense-aspect-modality combinations. DIPETT's performance on tense, modality, and negation phenomena is summarized in tables 5-4 through 5-7.

Table 5-3: DIPETT v3.0 — Summary of Results on TSNLP Phenomenon

TSNLP Phenomenon	*Number of Sentences*	*Fragmentary Parses*	*Correct Full Parses*	*Incorrect Full Parses*
C_Agreement-Collectives	2	0	2	0
C_Agreement-Coordinated_subj(correlatives)	12	6	6	0
C_Agreement-NP_V	2	0	2	0
C_Agreement-Parentheticals	6	3	3	0
C_Agreement-Partitives	17	11	5	1
C_Agreement-Pron_V	14	0	14	0
C_Agreement-Relative_clauses	7	1	7	0
C_Complementation-Divalent	43	1	41	1
C_Complementation-Equi	9	2	0	7
C_Complementation-Monovalent	2	0	2	0
C_Complementation-Raising	17	4	0	13
C_Complementation-Tetravalent	2	1	0	1
C_Complementation-Trivalent	75	2	44	29
C_Tense-Aspect-Modality	157	45	73	39
C_Negation-Tense_aspect_modality	287	126	90	72
C_Parentheticals-Embedded	1	1	0	0
NP_Coordination	135	0	135	0
NP_Modification-Relative_clauses	144	132	0	12
NP_Parentheticals	8	8	0	0
S_Parentheticals-Punct_interaction	3	3	0	0
S_Types-Questions-Wh	37	13	23	1
S_Types-Questions-Y/N	23	4	19	0
S_Types-Questions-Y/N_questions-Non_inverted-Non-tagged	14	14	0	0
Total	1017	377	466	176

5.4.2 Evaluation on TSNLP Phenomena and DIPETT Problems Identified

5.4.2.1 Clause Agreement Phenomena

The TSNLP group defines agreement as "the relationship between two grammatical elements such that if one of them contains a particular feature (plurality, for example) then the other also has to have that feature" (Estival *et al.*, 1995). The TSNLP documentation on agreement follows Quirk *et al.* for the most part.

Collectives

DIPETT does not recognize collective nouns as being plural, as discussed in 5.3.2.12 above.

Table 5-4: DIPETT v3.0 — Performance on TSNLP
Tense

Tense	*Number of Sentences*	*Correct Full Parses*	*Incorrect Full Parses*	*Fragmentary Parses*
simple past and simple present	2	2	0	0
aux + succeed	15	15	0	0
aux + succeeded	10	10	0	0
aux + succeeding	3	3	0	0
aux + be succeeding	13	12	1	0
aux + been succeeding	10	10	0	0
aux + have succeeded	13	10	0	3
aux + have been succeeding	13	10	3	0
aux + seen	3	1	2	0
aux + be seen	13	0	13	0
aux + been seen	10	0	10	0
aux + being seen	3	0	0	3
aux + be being seen	13	0	0	13
aux + been being seen	10	0	0	10
aux + have been seen	13	0	10	3
aux + have been being seen	13	0	0	13
Total	157	73	39	45

Table 5-5: DIPETT v3.0 — Performance on TSNLP
Tense plus negation

Tense	*Number of Sentences*	*Correct Full Parses*	*Incorrect Full Parses*	*Fragmentary Parses*
aux + *not* + succeed	37	29	5	3
aux + *not* + succeeded	4	3	0	1
aux + *not* + succeeding	4	4	0	0
aux + *not* + be succeeding	32	19	7	6
aux + *not* + been succeeding	4	3	0	1
aux + *not* + have succeeded	32	16	0	16
aux + *not* + have been succeeding	32	16	13	3
aux + *not* + seen	4	0	4	0
aux + *not* + be seen	30	0	24	6
aux + *not* + been seen	4	0	3	1
aux + *not* + being seen	4	0	0	4
aux + *not* + be being seen	32	0	0	32
aux + *not* + been being seen	4	0	0	4
aux + *not* + have been seen	32	0	16	16
aux + *not* + have been being seen	32	0	0	32
Total	287	90	72	125

Table 5-6: DIPETT v3.0 — Performance on TSNLP Modality

Auxiliary	*Number of Sentences*	*Correct Full Parses*	*Incorrect Full Parses*	*Fragmentary Parses*
can	8	1	3	4
could	8	4	2	2
could've	4	2	1	1
'd	12	6	3	3
did	1	1	0	0
does	1	1	0	0
had	4	2	1	1
has	4	2	1	1
is	3	1	1	1
'll	8	4	2	2
may	8	4	2	2
may've	4	2	1	1
might	8	4	2	2
might've	4	2	1	1
must	8	4	2	2
must've	4	2	1	1
ought to	8	2	2	4
's	7	4	1	2
shall	8	4	2	2
should	8	4	2	2
should've	4	2	1	1
used to	8	2	2	4
was	3	1	1	1
will	8	4	2	2
would	8	4	2	2
would've	4	2	1	1
Total	155	71	39	45

Correlation

DIPETT's recognition of correlative conjunctions is insufficient. For example, *both..and* is not recognized. Extending DIPETT's coverage of correlative conjunctions presents no problem. It should be noted that the coverage of correlative conjunctions in the TSNLP test suite is not complete either.

Table 5-7: DIPETT v3.0 — Performance on TSNLP Modality plus negation

Auxiliary	*Number of Sentences*	*Correct Full Parses*	*Incorrect Full Parses*	*Fragmentary Parses*
can not	8	1	3	4
cannot	8	1	3	4
can't	8	1	3	4
could not	8	4	2	2
couldn't	8	4	2	2
dare not	8	1	1	6
dared not	8	0	0	8
daren't	8	0	0	8
did not	1	1	0	0
did not use to	7	0	3	4
did not used to	8	0	4	4
didn't	1	1	0	0
didn't use to	8	0	4	4
didn't used to	8	0	4	4
does not	1	1	0	0
doesn't	1	1	0	0
had not	4	2	1	1
hadn't	4	2	1	1
has not	4	2	1	1
hasn't	4	0	0	4
is not	3	1	1	1
isn't	3	1	1	1
may not	8	4	2	2
mayn't	8	4	2	2
might not	8	4	2	2
mightn't	8	4	2	2
must not	8	4	2	2
Mustn't	8	4	2	2
need not	8	1	1	6
needn't	8	1	1	6
ought not to	8	2	2	4
oughtn't to	8	2	2	4
shall not	8	4	2	2
shan't	8	4	2	2
should not	8	4	2	2
shouldn't	8	4	2	2
used not to	8	2	2	4
usedn't to	8	0	0	8
was not	3	1	1	1
wasn't	3	1	1	1
will not	8	4	2	2
won't	8	4	2	2
would not	8	4	2	2
wouldn't	8	4	2	2
Total	287	125	90	72

Partitives

DIPETT does not recognize many partitives, as discussed in 5.3.2.10 above.

Pronoun Agreement

DIPETT generates a full parse tree for the following ungrammatical sentences:

* I are new.
* You am new.

5.4.2.2 Complementation Phenomena

In contrast to Quirk *et al.*, the TSNLP group defines *complementation* to include the subject as well as objects, prepositions, and obligatory adverbs.

Monovalent, Divalent

DIPETT's dictionary classifies monovalent (intransitive) and divalent (transitive) verbs, and DIPETT parsed these sentences very well. The only divalent phenomena that DIPETT failed to parse properly were the subjunctive (as noted above) and a sentence containing an intransitive phrasal verb.

DIPETT has provisions for phrasal verbs and the built-in dictionary contains many adverbial particle entries, but the feature is essentially "turned off" and not used. Preference is given to *nil* verbal particle; particles are almost always parsed as a simple adverb, and not as particles.

Trivalent

DIPETT did not perform well on the trivalent complementation tests. DIPETT does not recognize:

- Object complements as noted above in section 5.3.2.8.
- Obligatory adverbial complements and obligatory ditransitives as discussed in section 5.3.2.15.
- Complex transitives as shown in section 5.3.2.18.

- Transitive phrasal verbs, similar to intransitive phrasal verbs discussed above.

Sometimes DIPETT recognizes *wh*-complements, but not always; *wh*-complements may be misparsed as adverbials. DIPETT cannot distinguish between complementation patterns *obj+pp* and *iobj+pp*, but that would be difficult for a semantically weak parser to achieve.

Tetravalent

This is not a common complementation pattern; there are only two example sentences in the TSNLP data. DIPETT was unable to generate a full parse tree for one of those sentences "*She gives her the money for it*", and the other was not parsed as tetravalent "*She bets her one pound that he leaves.*" Complementation patterns are missing from DIPETT's code.

Equi and Raising

"Raising" of the object is not shown in DIPETT's parse trees, as discussed in section 5.3.2.18. Neither are the "equi" clausal complementation phenomena *Equi-Obj-control*, *Equi-Pobj-control*, and *Equi-Subj-control* shown in DIPETT parse trees. The tests for these phenomena, described by Soames and Perlmutter (1979), require that the referential subject of an infinitive or *to*-infinitive complement be identified. For example, this sentence tests the phenomenon *Equi-Obj-control*: "*Abrams urges Browne to hire Chiang.*" For this sentence, the object of the verb, *Browne*, is referentially the subject of the infinitive *hire*, and DIPETT's parse tree could reflect that:

S Abrams V urges O Browne C *to-inf* [$S_{equi\text{-}obj}$ Browne $V_{infinitive}$ hire O Chaing]

5.4.2.3 Diathesis

Active

The test items for this phenomenon are a subset of the *C_Complementation* test items described above.

Passive

DIPETT recognizes passive verb sequences only when the passive agent is specified. If the passive subject is *nil*, then the sentence is interpreted as being stative. See section 5.3.2.19.

Middle

This is the phenomenon of transitive verbs that have no passive correspondence, for example, "*He has a company.*" DIPETT is not aware of verbs that cannot be passivized. This was also identified by the Quirk test sentences.

Get as a passive auxiliary

DIPETT does not recognize *get* as a passive auxiliary, as discussed in section 5.3.2.22.

5.4.2.4 Coordination Phenomena

DIPETT was able to parse all of the noun phrase coordination examples. The grammatical test items for case assignment in the subject position contain a number of examples having conjoins in the accusative case. For example:

Me and him succeed.

DIPETT accepted all of these constructs and generated full parse trees.

5.4.2.5 Parenthetical Phenomena

DIPETT is extremely sensitive to punctuation and failed to parse all of the parenthetical test items. The TSNLP noun phrase parenthetical tests items are marked by square brackets, single dashes, and round brackets with a single comma. DIPETT can parse these tokens with some minor editing: round brackets instead of square brackets; double dashes instead of single dashes; remove the commas.

Some of the sentence parenthetical test items contain a sentence terminator such as a period or an exclamation mark within the parenthesis. DIPETT's scanner recognizes those as the

final sentence terminator; it terminates the sentence inappropriately; the sentence thus fails to parse.

5.4.2.6 Relative Clause Phenomena

DIPETT failed to parse most of the TSNLP test items for relative clauses. A few sentences that parsed fully were of the form "*The committee, the firm of which succeeds, comes.*" and parsed incorrectly having an appositive and not a relative clause .

All of the sentences that failed to parse have commas delineating the relative clauses, and the relative clauses are marked either by a relative pronoun or by a preposition followed by a relative pronoun. An experiment was performed to determine DIPETT's performance on those TSNLP relative clause test items with all of the commas removed. These results are more interesting and provide better diagnostic insight.

Relative clauses, with and without commas	*Number of Sentences*	*Correct Full Parses*	*Incorrect Full Parses*	*Fragmentary Parses*
NP_Modification-Relative_clauses	144	0	12	132
NP_Modification-Relative_clauses-no_commas	144	36	73	35

With commas removed:

- DIPETT correctly parsed all of the test items where the relative clause was introduced by a simple *wh*-word, for example, "*The committee which he accepts comes*".
- DIPETT parsed some but not all of the relative clauses introduced by a preposition an a *wh*-word marker, for example, "*The committee about which she argues succeeds*". The parse trees for the successful parses did not identify the preposition and *wh*-word as the relative clause marker.
- DIPETT failed to parse the remaining relative clause test items for reasons unrelated to relative clause phenomena. Most of the unsuccessfully parsed sentences presupposed collective noun agreement, and failed to parse because DIPETT does not recognize collective nouns as plural, for example, "*The committee who succeed come.*"

5.4.2.7 Questions

DIPETT's grammar for questions is deliberately simple. DIPETT was developed for the purpose of extracting knowledge from text, and it was not expected that much knowledge would be extracted from an interrogative statement. Still, DIPETT successfully parsed two thirds of the TSNLP question phenomena test items. DIPETT's performance on the TSNLP question phenomena was about the same as its performance on the whole TSNLP data.

5.4.2.8 Tense and Modality Phenomena

TSNLP provides very systematic testing of tense and modality phenomena. DIPETT should be able to parse all of these sentences correctly. Tables 5-4 and 5-5 summarize DIPETT's performance on tense phenomena; tables 5-6 and 5-7 summarize DIPETT's performance on modality. A number of problems were identified in DIPETT's verb sequence grammar.

The following problems in DIPETT were identified by the Quirk test sentences as well as the TSNLP sentences:

- DIPETT is unable to parse verb phrases containing forms of "*be being*" (see 5.3.2.17)
- DIPETT analyzes passive verb sequences as being stative (see 5.3.2.19)
- The marginal modal "*use to*" is sometimes not recognized (see 5.3.2.21)

The following problems were identified by the TSNLP suite but not by the Quirk sentences:

Dictionary and Tense Table are incomplete

DIPETT's internal dictionary includes closed class items such as auxiliaries. DIPETT's code also contains a tense table that converts DIPETT's internal tense notation to commonly accepted names. Neither the internal dictionary nor the tense table is complete. The dictionary is missing some entries for *can*, *dare*, and "*used to*". The tense table is missing some entries for tenses of *can*, *need,* and *dare.*

Scanner does not recognize some contractions

DIPETT's scanner is responsible for converting contractions to an uncontracted form. The scanner does not recognize the negative forms *daren't*, *usen't*, *usedn't*, and *hasn't*.

Marginal modal "dare"

Quirk *et al.* (1985; 3.42) state that the marginal modal *dare* exhibits abnormal time reference in that it can be used, without inflection for past as well as present time. Quirk *et al.* (1973; 3.54) show *dared* as the past form of the modal *dare*. The TSNLP suite contains sentences with the negated form *dared not*; DIPETT is unable to parse those sentences correctly.

5.4.3 Comparison of Evaluation Results on TSNLP and Quirk Sentences

TSNLP Sentences

DIPETT's parse trees were compared manually with the analysis provided in the TSNLP database. The TSNLP suite is very fine grained and pinpoints problems precisely. Phenomena are tested in isolation and not in combination, and phenomena coverage is not properly broad coverage.

Quirk Sentences

The Quirk test sentences are coarser grained than the TSNLP suite. They are broader in their coverage but not so precise in their diagnostic power. Even though only two chapters of Quirk *et al.* (1985) were used in the evaluation, the Quirk sentences broadly diagnosed most of the same problems that were identified by the TSNLP evaluation. The Quirk sentences also identified linguistic phenomena that are not covered by DIPETT and were not covered or diagnosed by the TSNLP sentences either.

6 Optimizing DIPETT

Try not to have a good time ... This is supposed to be educational.
-- Charles M. Schulz (1922 – 2000)

Following completion of the evaluation of DIPETT, modifications were made to the grammar of DIPETT v3.0 to produce a new DIPETT v4.0. This chapter describes the modifications to DIPETT's grammar and a re-evaluation to measure the improvement to DIPETT's performance on the Quirk and TSNLP sentences.

6.1 Defining "Optimize"

In this thesis, the word *optimize* is used in the common English sense of the word — "to make the most of" or "make best compromise between opposing tendencies". *Optimize* is not used in the computer science sense of "make faster" or " make most efficient use of resources".

6.2 Optimizing on TSNLP

Despite the fact that the Quirk sentences provided a good diagnostic evaluation of DIPETT's grammar, it was decided that the first goal for improving DIPETT should be to improve its performance on the TSNLP suite:

- As Table 5-1 shows, DIPETT performed worse on the TSNLP sentences than it did on the Quirk sentences. DIPETT generated a full parse tree for 77% of the grammatical Quirk sentences but only 65% of the grammatical sentences of the TSNLP suite.
- The TSNLP sentences are simple. Each TSNLP sentences covers a single isolated phenomenon; the Quirk sentences were deliberately chosen because they were more uncontrolled in their phenomena interaction.
- The TSNLP sentences are repetitive. Fixing one sentence often fixes several, and can cause a dramatic performance increase.
- Performance on TSNLP is a useful benchmark. Srinivas *et al.* (1998) published the performance of XTAG on TSNLP; XTAG parses 61.4% of the grammatical sentences.

Had DIPETT's performance on TSNLP been better, then it would have been more interesting to optimize DIPETT on the Quirk sentences.

6.3 Modifications to DIPETT

Natural language parsers are notoriously tricky, especially those written in Prolog. Parsing is also very sensitive to deviations in the lexicon. DIPETT is implemented in Quintus Prolog; the actual logic grammar of the parser is 600 DCG rules in 3200 lines of Prolog, and the built-in dictionary contains over 3000 entries.

Modifications to DIPETT were made as unobtrusively as possible to minimize the risk of introducing errors and side effects.

6.3.1 Correlative Conjunctions

6.3.1.1 "Both...and"

The correlative conjunctions allowed in DIPETT's grammar were insufficient. The noun phrase grammar allowed only "either...or" and "neither...nor", but the TSNLP clausal agreement phenomena had several sentences containing the correlative pair "both...and". The following rule was added to DIPETT's grammar, cloned from the rules for "either...or" and "neither...nor":

```
/* BOTH/AND Nps.          [EAS: added both..and]
*/
either_or__np(POS,conj_nps(both_and,NUM,[Np_e,Np_o]),NUM) -->
      conj(both), np_nucleus_and_mod(POS,Np_e,Num_e),
      conj(and), np_nucleus_and_mod(POS,Np_o,Num_o),
      {plur_rule_n(Num_e,Num_o,and,NUM)}.
```

6.3.1.2 Additional correlatives required

The addition of "both..and" fixed the immediate problem with the TSNLP *C_Agreement* phenomenon, but further additions should be made for DIPETT to be considered truly broad coverage:

- Quirk *et al.* (1985; 13.33-42) lists "not (only)...but (also)" as a correlative.

- Similar additions should be considered for every part of speech, not just noun phrases. For example, as a verb: "*He not only protested but also refused to pay his taxes.*"
- DIPETT recognizes some forms of "neither..nor" but fails to parse the sentence "*He neither fished nor cut bait.*"

6.3.2 The Passive Voice

6.3.2.1 Passives not parsed

Delisle's original implementation of DIPETT parsed passive verb sequences correctly, but DIPETT v3.0 was unable to parse any of the TSNLP test items for the passive tense. The parse trees that were produced were all stative and not passive, and had the past participle functioning as an adjective complement. For example:

Sentence	*DIPETT v3.0 interpretation*	*DIPETT v4.0 interpretation*
He was seen.	✘ S He $V_{copular}$ was C *adj* seen	✔ $V_{passive}$ was seen O he
He was green.	✔ S He $V_{copular}$ was C *adj* green	✔ S He $V_{copular}$ was C *adj* green

It appears that this problem was introduced by an unsuccessful attempt to add parsing of progressive passives to DIPETT's grammar, for example, *"He was being seen."*

DIPETT's grammar for passive verbs was restored to Delisle's original implementation, but updated to conform with the current convention for passing tense parameters. This modification allowed DIPETT to parse some passives, but did not itself permit parsing of the above simple passive sentence because the passive subject (agent) is *nil*.

6.3.2.2 "nil" Passive Subject

DIPETT's parse tree output for passive sentences identifies the *active subject* and *active object* relating to the active verb form. Quirk *et al.* (1985; 3.65) define the active-passive correspondence for monotransitive verbs:

Sentence	*Voice*	*Active-Passive correspondence*	*DIPETT interpretation*
She sees him.	Active	NP_1 + V_{active} + NP_2	**S** She V_{active} see **O** him
He is seen by her.	Passive	NP_2 + $V_{passive}$ + *by* NP_1	$S_{passive}$ Her $V_{passive}$ see **O** he

DIPETT's verb complement grammar never did allow for the *nil* passive subject of **SVO** structures. Passive **SVO** structures with *nil* passive subject were added to DIPETT's list of possible transitive complements. The transitive object is the passive subject, and does not follow the verb. DIPETT's grammar was made to accept an empty object following the passive verb:

```
complement_tr(passive, nil, svo([pass_subj(nil), nil]))  --> [].
```

This addition enabled DIPETT to parse many of the passive TSNLP test items for low level tense-aspect-modality phenomena. At that point, problems in the tense table and with *being* prevented the remaining passive tense-aspect-modality test items from parsing successfully.

6.3.3 The Progressive Form of *be*

DIPETT v3.0 did not recognize *being* as a progressive form of the *be* auxiliary. This omission had two main effects:

- All sentences containing passives in the form "*be being*" failed to parse, *e.g.*, "*I am being seen.*"
- All sentences containing statives in the progressive failed to parse, *e.g.*, "*I am being good.*"

First, an entry was made to DIPETT's dictionary for *being* as a progressive form of the auxiliary verb *be*. DIPETT's Lexicalyser also required an entry for *being*[20]. These additions allowed the parsing of progressive passives involving *being*.

[20] So that *get_num_aux*, the predicate that returns the number (1st, 2nd, or 3rd person), was forced to return the number *Num* instantiated.

Then, *being* was added to DIPETT's dictionary as a progressive stative verb. That allowed DIPETT to parse sentences such as "*I am being good.*" but had an unexpected side effect. DEVAL's report showed that all of the perfective progressive passive sentences "*have been being*", which had been parsing correctly, were then parsed as **SVOC** structures with the main verb *have*, object "*been being*", and complement *seen*. The sentence "*I have been being seen.*" was analyzed as:

S | **V** have **O** *NP* [*adj* been *ing-noun* being] **C/A** *adj* seen

The grammar rules responsible for that interpretation are:

- A past participle may function as an adjective (Quirk *et al.*, 1985; 7.15)
- A nominal *–ing* clause may function as direct object (Quirk *et al.*, 1985; 15.12).

Adding *being* as a progressive thus permits its use as a nominal.

One solution to this problem might be to specifically disallow *being* to be used as a nominal. It was decided instead to block the past-participle *been* from functioning as an adjective. Quirk *et al.* (1973; 5.46) states that the *–ed* participle of verbs can be used as attributive adjectives. However, *been* is never used as an adjective. Blocking *been* as an adjective did not cause any further problems, and the parse trees for the perfective passive statives were restored. The sentence "*I have been being seen.*" is analyzed as:

S *pass-subj* nil **V**passive have been being seen **O** |

6.3.4 The Marginal Modal "*used to*"

The grammar for DIPETT v3.0 does not always properly parse sentences containing the marginal modal "*used to*", as discussed in section 5.3.2.21.

6.3.4.1 "used to" in the passive and stative

Given a passive or stative sentence involving "*use to*" DIPETT finds and accepts an incorrect active interpretation with the main verb *use* and with *to* as a *to*-infinitive marker or as a preposition.

S He V used O *nom-cl* [*to-inf* $V_{copular}$ be C *adj* seen]
S He V used O *nom-cl* [*to-inf* $V_{copular}$ be C *adj* good]

That specific problem was fixed by checking for passive and stative interpretations of "*used to*" before searching for an active interpretation.

6.3.4.2 "used to" in aspect and tenses

DIPETT v3.0 recognizes "*used to*" but not "*use to*" and does not recognize either with DO-support. Quirk *et al.* (1985; 3.44) state that the *used* marginal modal always takes the *to*-infinitive and occurs only in the past tense, and that "*use to*" and "*used to*" occur interchangeably with DO-support as in "*He didn't use(d) to smoke.*" They also state that "*did ... used to*" is not considered standard.

Additions were made to DIPETT's dictionary for "*used to*" and "*use to*" in the past tense, and DO-support was added to the verb sequence grammar.

6.3.5 Missing Tenses in Combination with Modal Auxiliaries

DIPETT's code contains a tense table that converts DIPETT's internal tense notation to commonly accepted names. The tense table for DIPETT v3.0 is missing entries for tenses of *can, need, dare*, and "*used to*". The built in dictionary does not have entries for *can* and *dare* in the perfective or for the past tense of *dare*.

Entries were added so that DIPETT v4.0 could parse all of the TSNLP tense-aspect-modality test items. These items include some that contain the past form of the *dare* modal appearing as "*dared not*", for example, "*She dared not be seen.*" The two Quirk grammars contradict themselves as to the grammaticality of *dared*, as discussed in section 5.4.2.8. It was decided to allow it.

No grammatical sentence should fail in the tense table. A predicate was added so that the tense table constructs a tense label if an entry is missing. It is unlikely that that predicate will ever execute, as the TSNLP suite is very fine grained and rigorous in its testing of combinations of auxiliaries and tenses. The automatic tense label construction predicate was disabled for the final evaluation of DIPETT v4.0. Experiments should be performed to test the effect of allowing prepositional phrases as adverbs.

6.3.6 Missing Negative Contractions

DIPETT v3.0 does not recognize negative contractions *daren't*, *usen't*, *usedn't*, and *hasn't*. These contractions were added to the scanner[21].

6.3.7 Passive Auxiliary "*get*"

DIPETT v3.0 does not recognize *get* as a passive auxiliary or as a copular verb. Entries were made to DIPETT's internal dictionary so that all forms of *get* are recognized as both passive and stative. DIPETT now treats *get* similarly to the *be* auxiliary.

DIPETT v4.0 successfully parses *get* as described by Quirk *et al.* (1985; 3.66)

Sentence	*Verb form*	*DIPETT v4.0 interpretation*
I got the tickets.	Transitive	S ! $V_{active\text{-}past}$ get O the tickets
I am getting very old.	Stative	S ! $V_{copular\text{-}pres\text{-}continuous}$ get C very old
I got corrected by him.	Passive	$S_{passive}$ Him $V_{passive\text{-}past}$ correct O !

6.3.8 Adverbial Complements

6.3.8.1 Optional Adverbials

The tetravalent pattern for the TSNLP sentence "*She gives her the money for it.*" does not exist in DIPETT v3.0 and the sentence fails to parse. The simplest and most immediate way to fix that problem is to add the tetravalent complementation pattern **SVOOA** to DIPETT's complements. That was done, and the above TNSLP sentence now parses.

[21] Thanks to Stan Szpakowicz for pointing out the coding error "hasen't".

A better and more general solution would be to address the problem discussed in section 5.3.2.4. that DIPETT does not always recognize prepositional phrases as being adverbs. If DIPETT did recognize prepositional phrases can function as adverbs, then the above sentence would be parsed as an **SVOO** clause type with an optional subordinate final adverbial, and a specific entry for **SVOOA** complement would be unnecessary. That solution is preferable because Quirk *et al.* (1985; 2.15) show that multiple optional adverbials can occur, particularly in the final position. It is neither desirable nor practical to enumerate each possible combination in the code for complements.

6.3.8.2 Obligatory Adverbials

In their seven basic clause types, Quirk *et al.* (1985; 2.16) define two obligatory adverbial clause types: **SVA** for transitive verbs; **SVOA** for intransitive verbs.

SVOA

DIPETT v3.0 is able to parse sentences that have an active transitive verb with complements *o+pp*, but the parse trees that it generates do not differentiate between the **SVOO**, **SVOC**, and **SVOA** clause types.

SVA

DIPETT v3.0 is not able to parse sentences that have an active intransitive verb and prepositional phrase complement; there is no rule for obligatory adverbial intransitive **SVA** structures. Since **SVA** is one of the fundamental clause types, this is a not the same issue as the optional adverbials discussed above. A rule was added to DIPETT v4.0 to allow **SVA** structures having a prepositional phrase:

```
complement_intr(active, nil, sva(X))                    -->
   required_prepphrases(X, _), {!}.
```

This rule allows DIPETT v4.0 to parse sentences such as "*He depends on me.*"

6.3.9 Relative Clauses

DIPETT failed to parse most of the TSNLP relative clause test items, as discussed in section 5.4.2.5.

6.3.9.1 Commas

Modifications were made to the rules for noun postmodifiers and to the rules for heuristic lookahead for relative clauses. Both allow relative clauses to be delimited by commas when the relative clause is marked by a relative pronoun or by a preposition followed by a relative pronoun.

6.3.9.2 Ordering of Heuristic Lookahead Rules

Relative clauses may be introduced by a preposition followed by a *wh*-word. An example of a sentence containing such a relative clause is: "*The firms with which she deals succeed.*" DIPETT v3.0 has a heuristic lookahead rule that specifies that a preposition followed by a relative pronoun looks like a relative clause.

Prolog code is very sensitive to the ordering of rules. In DIPETT v3.0 the heuristic rule that identifies prepositional phrases executes first, before the heuristic rule that identifies relative clauses. More specifically, this occurs only for prepositional phrases modifying nouns in the pre-verb position, but almost all of the TSNLP noun phrase modification test items are like that. DIPETT v3.0 produces parse trees for sentences such as the above example as though the preposition marks a noun phrase post-modifier, which is a prepositional phrase consisting of a preposition and a noun phrase nominal *wh*-clause.

The heuristic rules were re-ordered in DIPETT v4.0. A preposition followed by a relative pronoun looks like a relative clause, and following that rule is one that states that a preposition "looks like" a prepositional phrase.

6.3.9.3 Parse Tree Output

For the example sentence "*The firms with which she deals succeed.*" the relativizer is the preposition followed by a *wh*-word relative pronoun. DIPETT's parse tree output has been modified to reflect that:

```
[...]
np_postmodifiers
  rel_clause
    rel_marker  with  which
    [...]
```

6.3.10 The Mandative Subjunctive

The grammar for DIPETT v3.0 does not recognize any of the subjunctive forms. The mandative subjunctive is quite straightforward. Quirk *et al.* (1985; 3.59) define the mandative subjunctive as occurring in subordinate *that*-clauses and consisting of the base form of the verb only.

A rule was added to DIPETT v4.0 that allows a full *that*-clause to take an infinitive verb sequence:

```
full_that_clause(full_that_clause(CL))                    -->
conj(that), sentence_core_that(CL).
sentence_core_that(Struct)                                -->
np_equiv(NP), verbphrase([inf], NP, Struct).
```

These rules allow parsing of the mandative subjunctive, but need further consideration before being accepted as a final solution:

- The full_that_clause predicate is invoked only through nominal_clause and so, without further modifications, the parse trees show the mandative subjunctive clause as a nominal object. For example, the sentence "*They demanded that she call him.*" is parsed:

  S they V demanded O *nom-cl* [*full-that-cl* S she V call O him]

- Quirk *et al.* (1985; 3.59, 16,32, 16.72) lists specific verbs, adjectives, and nouns that commonly introduce a *that*-clause containing the mandative subjunctive. The grammar should restrict the use of the mandative subjunctive to those nouns, verbs, and adjectives.

6.3.11 Rule for Noun Conjunction Number

DIPETT v3.0 rule *plur_rule_n* for determining whether a noun phrase is singular or plural states:

- A noun phrase in the form "*X or Y*" is singular if either of *X* or *Y* is singular.
- A noun phrase in the form "*X or Y*" is plural if either of *X* or *Y* is plural.

These rules are incorrect and inconsistent, and so DIPETT v4.0 has:

- A noun phrase in the form "*X or Y*" is singular if *Y* is singular.
- A noun phrase in the form "*X or Y*" is plural if *Y* is plural.

This modification permits DIPETT v4.0 to correctly parse such TSNLP sentences as "*Either they or he is new*" and to reject sentences such as "*Neither they nor she are new.*"

6.3.12 Declarative Sentences Terminated by Question Mark

DIPETT v3.0 recognizes declarative and imperative sentences, and interrogative sentences terminated by a question mark. The TSNLP phenomenon *S_Types-Questions-Y/N_questions-Non_inverted-Non-tagged* has a number of test items that are simple declarative sentences but terminated by a question mark, for example, "*He can carry it?*"

DIPETT labels parse trees for declarative and imperative sentences with sentence type *parse_tree__decla_or_imper*, and parse trees for interrogative sentences with type *parse_tree__inter*. A new sentence type was created for DIPETT v4.0 so that the test items of this phenomenon could be parsed. The parse trees for these sentences are labelled *parse_tree__inter_decla_or_imper* to show the uncertain status of these sentences.

6.3.13 Stative SVOC and SVOA

DIPETT's grammar allowed for **SVC** and **SVA** clause types with stative verbs, but not the related clause types for transitive verbs. Rules for the **SVOC** and **SVOA** clause types for stative verbs were added to DIPETT's grammar:

```
complement_stat(stative, svoc_svoa([Obj, A]))          -->
   np_equiv(Obj),
   compl_adj_phrase(A).

complement_stat(stative, svoc_svoa([Obj, A]))          -->
   np_equiv(Obj),
   np_equiv(A).
```

6.4 Comparative Evaluation of the Two Versions of DIPETT

6.4.1 Summary of Parse Results

Table 6-1 summarizes the results of parsing both test suites with both parsers. The modifications to DIPETT's grammar were intended to improve DIPETT's performance on the TSNLP suite, as discussed in section 6.2, and a marked improvement in the number of full parses generated by DIPETT has been achieved — an increase from 65% to 92% full parses. The same improvements also cause a more modest improvement in the coverage of the Quirk sentences, even though no attempt was made to "tune" DIPETT to that test suite. The number of full parses on all of the 578 Quirk sentences rose from 79% to 85%.

Little improvement is shown in parsing (rejecting) ungrammatical sentences, but no attempt was made to optimize DIPETT for that purpose. Some grammar additions actually permit DIPETT to successfully parse ungrammatical sentences that were not parsed before. This condition is not entirely due to shortcomings in DIPETT's grammar; the ungrammatical sentences in the test suites must be pruned to those that are properly ungrammatical for a surface syntactic parser before meaningful results for ungrammatical sentences could be achieved. The ungrammatical test items for the TSNLP complementation phenomena need not be used; neither should the Quirk sentences that are semantically but not syntactically incorrect.

6.4.2 Comparison of Parse Results

6.4.2.1 DEVAL Sentence Comparison

Table 6-2 shows DEVAL's classification of the comparison of the parse trees generated by the two versions of DIPETT. The results that need to be examined fall in two categories:

- The sentences that were fully parsed by the new version of DIPETT but not the old.
- The sentences that were fully parsed by both parsers, but the parse trees are different.

Table 6-1: Comparison of Total Number of Sentences Parsed

Grammatical Sentences	Quirk *et al.* Chapter			TSNLP
	2	3	5	
Number of grammatical sentences	252	308	18	1143
Number of full parses generated by DIPETT v3.0	218 (86%)	221 (72%)	18 (100%)	745 (65%)
Number of full parses generated by DIPETT v4.0	226 (90%)	248 (81%)	18 (100%)	1045 (92%)

Ungrammatical Sentences	Quirk *et al.* Chapter			TSNLP
	2	3	5	
Number of ungrammatical sentences	22	44	10	1519
Number of fragmentary parses generated by DIPETT v3.0	7 (32%)	17 (39%)	5 (50%)	800 (53%)
Number of fragmentary parses generated by DIPETT v4.0	6 (27%)	15 (34%)	5 (50%)	719 (47%)

Table 6-2: Comparison of Parse Results for DIPETT v3.0 and DIPETT v4.0

Grammatical Sentences	Quirk *et al.* Chapter			TSNLP
	2	3	5	
Not successfully parsed by either DIPETT	26 (10%)	57 (19%)	0 (0%)	88 (8%)
Parsed fully *only* by DIPETT v4.0	8 (3%)	30 (10%)	0 (0%)	310 (27%)
Parsed fully *only* by DIPETT v3.0	0 (0%)	3 (1%)	0 (0%)	1 (0%)
Both have full parse trees, the parse trees are identical	210 (83%)	189 (61%)	18 (100%)	599 (52%)
Both have full parse trees, the parse trees are not the same	8 (3%)	29 (0%)	0 (0%)	145 (13%)
Total	252	308	18	1143

Ungrammatical Sentences	Quirk *et al.* chapter			TSNLP
	2	3	5	
Not successfully parsed by either DIPETT	6 (27%)	15 (34%)	5 (50%)	651 (43%)
Parsed fully *only* by DIPETT v4.0	1 (5%)	2 (5%)	0 (0%)	192 (13%)
Parsed fully *only* by DIPETT v3.0	0 (0%)	0 (0%)	0 (0%)	2 (0%)
Both have full parse trees, the parse trees are identical	15 (68%)	24 (55%)	5 (50%)	595 (39%)
Both have full parse trees, the parse trees are not the same	0 (0%)	3 (7%)	0 (0%)	79 (5%)
Total	22	44	10	1519

6.4.2.2 Change in DIPETT's Performance on the TSNLP Sentences

Appendix E shows the results of DIPETT v4.0 parsing the TSNLP test items. Table 6-3 shows a summary of the changes in DIPETT's performance on the TSNLP phenomena. The data included in Table 6-3 is only for the sentences where a full parse tree changed, or a full parse tree was generated for a sentence where there had been a fragmentary parse.

The most notable improvements in the performance of DIPETT v4.0 on the TSNLP test items are:

- Correlative Conjunctions: all of the test items parse correctly.
- Tense and Modality: all of the tense and modal combinations generate full parses.
- Passive Voice: passive voice is identified properly as being passive, and not stative.
- Relative Clauses: all of the test items generate full parses.
- Other additions cause minor changes in the overall score on TSNLP: the subjunctive, passive *get*, tetravalent complementation, *being* as a progressive stative.

6.4.2.3 Change in DIPETT's Performance on the Quirk Sentences

Appendix F shows the results of DIPETT v4.0 parsing the Quirk test sentences. Thirty-eight new parse trees are generated, and thirty-three parse trees have changed. Almost all of these changes are improvements — most of the newly generated parse trees are acceptable, and the

parse trees that are different have changed for the better. Still, not all the parse trees that have changed are completely correct.

Table 6-3: Summary of Change in Performance on the TSNLP Phenomena

Phenomenona	Number of Parses Changed	DIPETT v3.0			DIPETT v4.0			Improvement in	
		Fragmentary	Correct Full Parses	Incorrct Full Parses	Fragmentary	Correct Full Parses	Incorrect Full Parses	Number of Full Parses	Number of Correct Parses
C_Agreement-Coordinated_subj(correlatives)	6	6	0	0	0	6	0	6	6
C_Complementation-Divalent	1	1	0	0	0	1	0	1	1
C_Complementation-Trivalent	1	1	0	0	0	1	0	1	1
C_Complementation-Tetravalent	1	1	0	0	0	1	0	1	1
C_Diathesis-Active	3	3	0	0	0	3	0	3	3
C_Diathesis-Passive	22	12	1	9	0	22	0	12	21
C_Negation-Tense_aspect_modality	199	125	2	72	0	199	0	125	197
C_Tense-Aspect-Modality	86	45	1	40	0	86	0	45	85
NP_Modification-Relative_clauses	112	99	1	12	0	106	6	99	105
S_Types-Questions-Wh	5	5	0	0	0	0	5	5	0
S_Types-Questions-Y/N	5	1	4	0	0	4	1	1	0
S_Types-Questions-Y/N_questions-Non_inverted	13	13	0	0	0	13	0	13	13
Total	454	312	9	133	0	442	12	312	433

The improvements in the performance of DIPETT v4.0 on the Quirk test sentences are:

- Tense and Modality: all of the tense and modal combinations generate full parses.
- Passive Voice: passive voice is identified properly as being passive, and not stative.

- The subjunctive parses correctly.
- The verb *get* is recognized as a passive.
- *being* is recognized as a progressive stative verb.
- "*used to*" parses correctly as a marginal modal.

6.4.2.4 Side Effects and Remaining Problems

Some of the additions to DIPETT's grammar had interesting and unexpected side-effects. While all full parse trees generated by DIPETT v4.0 have been verified and found perfectly adequate, a few small deficiencies can be noted in sentences for which the grammar was not specifically optimized.

Relative Clauses

Almost all of the TSNLP relative clause test items parsed successfully. Those that did not were the test items for the non-restrictive genitive phenomena. The sentence "*The committee, whose firm succeeds, comes.*" should be parsed with clauses:

"*The committee comes*" and "*The committee's firm succeeds*"

DIPETT v4.0's parse tree shows, however, that:

"*The committee comes*" and "*The firm succeeds of the committee*".

DIPETT's relative clause grammar needs to be extended to allow for genitives.

Questions

Addition of the parse tree type *parse_tree__inter_decla_or_imper* to parse some of the TSNLP test items was discussed in section 6.3.12. That grammar change worked well for some of the TSNLP sentences, but other sentences (in both test suites) were parsed unintentionally. The implementation of the grammar addition is not restrictive enough. Two of the problems identified with the implementation of *parse_tree__inter_decla_or_imper* are:

- *Wh*-questions may look like declarative sentences with a *wh*-word as a determiner. The Quirk sentence "*Which cup is yours?*" is parsed similarly to "*That cup is yours.*"
- Sentences may be parsed as imperative *parse_tree__inter_decla_or_imper*, which is rather meaningless. The parse tree for the sentence "*Have you a box of matches?*" is similar to that for "*Have yourself a Merry Christmas!*"

If DIPETT's "question grammar" were stronger, it would parse these examples itself, before the *parse_tree__inter_decla_or_imper* rule is ever encountered. Regardless, it is not clear that *parse_tree__inter_decla_or_imper* is a very useful addition outside of parsing the TSNLP suite. The rule could be abandoned, but if it is kept, then it should be modified to allow only sentences that meet the strict TSNLP criteria: yes-no statements that are not inverted and not tagged.

Tense Labels

DIPETT v4.0 generates full parse trees for all of the TSNLP tense test items and all of the Quirk sentences relating to verb tense sequences. The tense labels need to be examined closely. A filter was written to examine the tense labels in parse trees, and it identified several shortcomings:

- The tense labels are not consistent. For example, the simple present is shown as *present_simple* with some modals but *present* with others.
- The *could*, *would*, and *should* conditionals label the perfect aspect as being *simple past*.
- The modal "used to" does not indicate the continuous aspect.

6.5 Efficiency of DIPETT

Delisle (1994) and Barker *et al.* (1998) showed that DIPETT delivers a parse analysis in a relatively short time, if it is going to at all. The parse runs on the Quirk and TSNLP sentences confirm this result. Most of the Quirk and TSNLP sentences are parsed promptly.

Excessively long parse times were observed on only a few sentences, and those contained constructions that are not covered by DIPETT's grammar.

A simple improvement to DIPETT's parsing efficiency of verb sequences should be made. Rejection of verb sequences because the tense is not acceptable is performed late, even when the acceptable tenses are known in advance. The sentence "*I go.*" presents an example in the case of the simple present. In parsing this sentence DIPETT's top level sentence rule

```
S --> NP VP
```

fails twice before it succeeds, because its verb phrase code returns two nonfinite analyses (*imp* and *inf*) of the verb *go* before suggesting the simple present. If the tense must be restricted to one of a known list, then this information should be passed into the verb sequence handler.

7 Comparing DIPETT to Online Parsers

Empirical assessment of practical parser performance has become an established technique and continues to be the primary means of comparison among parsing algorithms. Two parsers can only be compared meaningfully when they use the same grammar or achieve demonstrably similar competence on the same test set.

In 1999, several natural language parsers are available through the Internet, and their links are listed on some of the common NLP repository sites. Is DIPETT ready to be similarly advertised? This chapter describes a small experiment to compare DIPETT's capabilities with those of some online parsers.

7.1 Search for Parsers

7.1.1 Parsers Available Through the Internet

Parsers on the Internet were found through three main sources: Martin Volk[22] at the University of Zurich maintains a list of online computational linguistics tools; Kenji Kita[23] at Tokashima University in Japan maintains a page of speech and language web resources including software tools for NLP; the Association for Computational Linguistics provides the NLP/CL Universe[24] search engine for NLP and CL Web sites.

A number of parsers were located. Some have a Web interface that allows one to enter a sentence and have their parse displayed; other parsers must be downloaded and installed. The software programs investigated in this chapter are specifically listed as parsers, not part-of-speech taggers.

The following parsers have online demonstrations available:

1. Natural Language Processing Parser Demonstration Page (Georgetown University)
2. Natural Language Parser Demo (Alpo Lind, Finland)

[22] Interactive Online CL Demos — http://www.ifi.unizh.ch/CL/InteractiveTools.html

[23] Software Tools for NLP — http://www-a2k.is.tokushima-u.ac.jp/member/kita/NLP/nlp_tools.html

3. EngLite Parser — Functional Dependency Grammar Parser (Conexor, Helsinki)
4. ENGCG — Constraint Grammar Parser of English (Lingsoft, Helsinki)
5. The Link Parser — A parser and grammar for English (Carnegie Mellon University)
6. Head-corner parser applied to Alvey Natural Language Tools grammar (University of Groningen)
7. IPS — Interactive Parsing System (University of Geneva)

Two online parsers are listed but were not accessible:

1. PRINCIPAR — A Broad-coverage Principle-based Parser (University of Manitoba)
2. LINGO — Linguistic Grammars Online — A multi-purpose broad-coverage grammar of English (Stanford University)

Other parsers have no web-based user interface and must be downloaded:

1. PC-PATR — A syntactic parser (Summer Institute of Linguistics)
2. Apple Pie Parser — A bottom-up probabilistic chart parser (New York University)
3. Attribute-Logic Engine (ALE) System and Grammars (Carnegie Mellon University)
4. PAPPI — A principle-based parser

Appendix H contains the Internet addresses for all of the online parsers visited.

7.2 Selection of Parsers for Comparison

7.2.1 Initial Tests and Observations

An initial test of two sentences was performed on each online parser visited; the test sentences chosen were "*The man upon whom I depended disappeared.*" and "*The cat has eaten the canary and I am devastated.*" DIPETT is able to parse both of those sentences correctly. The parser output was examined to determine:

[24] The NLP/CL Universe — http://www.acl.web.org

- Whether the sentences had been successfully parsed.
- If part of speech tagging had been performed.
- If tense, voice, and internal clause and phrase structures had been identified.
- If one or more than one parse had been generated.

Table 7-1 shows the performance of the online parsers on the initial test.

7.2.2 Parser Selection

It was decided to use the Link Parser, IPS, and PC-PATR for comparison with DIPETT. Link and IPS can be accessed online, their output describes parse tree structures (not dependency based and not simple taggers) and they successfully parsed the initial two test sentences. PC-PATR is not an online parser, and must be downloaded and installed, but its parse tree output is easy to read and clearly shows constituent phrases.

Table 7-1: Initial Test of Online Parsers

Parser	*Purpose / Grammar*	*Generates Multiple Parses?*	*Parse Tree Identifies*			*Parses the two test sentences ?*
			Part of Speech	*Tense Voice*	*Clauses Phrases*	
Georgetown NLP Demo	GB grammar	No	Yes	No	Yes	No
Groningen Head-corner Parser	Alvey grammar	No	Yes	No	Yes	No
Alpo Lind's NLP Demo	LSP [25] syntax definition	Yes	Yes	No	Yes	No
EngLite FDG Parser	Dependency analysis	No	Yes	No	Yes	Yes
ENGCG	Morphological analysis	Multiple POS tags	Yes	Tense	No	Yes
Link Syntactic Parser	Link grammar	Yes	Yes	Tense	Yes	Yes
IPS	GB grammar	No	Yes	No	Yes	Yes

7.3 Parser Comparisons

7.3.1 Choice of Test Sentences

Sixty-one sentences were selected; the selections represented a broad range of the TSNLP phenomena that were also covered by the Quirk test sentences. The linguistic phenomena tested were: active and passive verb sequences, modality including negation and contractions, premodification and quantification, correlative conjunction, *get* as a passive auxiliary, complementation, relative clauses, nominal clauses, postmodification and apposition.

7.3.2 Summary of Parser Performance[26]

Table 7-2 and Table 7-3 show the parse results of the test sentences. The parses of the grammatical sentences were evaluated very simply. A parse was considered acceptable if:

- a full parse tree was generated.
- its part-of-speech tags were correct.
- the clause structure, phrases, linkages or bracketings were plausible.

[25] The definitions of English syntax made by Linguistic String Project. http://cs.nyu.edu/cs/projects/lsp/

[26] Terry Copeck installed PC-PATR, and parsed all of the test sentences on all three parsers. Grateful thanks to Terry for his help and also to Stan Szpakowicz for funding his assistance.

Table 7-2: Results of Parser Comparison Experiment – Grammatical Sentences

Phenomenon	*Number of Sentences*	*Acceptable Parses*			
		PC-PATR	Link	IPS	DIPETT v4.0
Active verb sequences	8	4	8	8	7
Correlative conjunction	7	0	2	4	6
get as a passive auxiliary	1	0	1	0	1
Modals	15	0	13	3	13
Nominal clauses	4	0	3	4	4
Passive verb sequences	7	2	7	4	7
Postmodification and apposition	9	1	2	3	2
Premodification and quantification	9	0	7	4	2
Relative clauses	6	0	5	6	6
Total	66	7 (11%)	48 (70%)	36 (50%)	48 (70%)

Table 7-3: Results of Parser Comparison Experiment – Ungrammatical Sentences

Ungrammatical Sentences	PC-PATR	Link	IPS	DIPETT v4.0
Number of Sentences Tested	—	73	72	73
Number of Full Parse Trees Generated	—	18	46	42

7.3.2.1 PC-PATR

PC-PATR was difficult to install. Pre-processing of the text was required before any parses were successful: downshifted initial capitals, removed all punctuation, expanded contractions, replaced personal names with ones known in the lexicon.

Even then, PC-PATR did not perform well. It is not supplied with a full-fledged grammar. Probably this parser is intended to be used with a user-supplied grammar, and possibly it is more capable than our experiment indicates. There are two morphological analyzers built in

(PC-KIMMO and AMPLE) and the 20k-word lexicon contains contractions that the parser failed to recognize.

7.3.2.2 Link, IPS, and DIPETT

The Government & Binding grammar of IPS is rather non-specific, and its parse trees are not as detailed as DIPETT's or the Link Parser's. IPS generated approximately 40% fragmentary parses on the 61 test sentences.

The output of the Link Parser is difficult to read, but it appears to perform about as well as DIPETT on the 61 test sentences. The Link Parser developers acknowledge that their grammar performs poorly on correlative conjunctions.

7.4 Other Tests

7.4.1 Testing the Link Parser on the Quirk Test Sentences

The Link parser was tested on 436 of the grammatical Quirk test sentences extracted for this thesis; it was able to generate at least one complete parse tree for 87% of those. How many of the "first" parse trees are correct is being investigated.

7.4.2 Testing DIPETT on the Link Test Sentences

The Link Parser provides a suite of test sentences that it claims to parse 100% correctly. These sentences were downloaded and parsed using DIPETT v4.0; DIPETT v4.0 generated full parses for 65% of the unedited sentences.

This seemingly low score does not necessarily reflect weaknesses in DIPETT's grammar. Many of the Link sentences contain punctuation, particularly quotations and abbreviations. These are known weaknessness of DIPETT and outside the scope of this thesis, which was mainly syntax and not preprocessing.

Further work will be undertaken to determine how many of the Link test sentences that ought to be covered by DIPETT's grammar actually do generate a correct parse tree.

7.4.3 Testing DIPETT on the Alvey Test Sentences

The purpose of this experiment was to compare DIPETT's performance with other parsers freely available through the Internet. The Alvey Natural Tools Parser can be acquired but cannot be used, even for research purposes, without payment of a licensing fee. The University of Groningen site makes available their Head-corner parser applied to Alvey Natural Language Tools grammar. A sample set of sentences for the Alvey grammar, including parse trees generated by the parser, are available from that site. These were downloaded and the declarative sentences were parsed with DIPETT. DIPETT generated full parse trees for 56% of those sentences.

It should be noted that many of the sentences are deliberately convoluted and extremely awkward, and do not contain punctuation to mark the phrases. DIPETT relies on punctuation to mark some noun postmodifiers, and is as sensitive to its absence as its unexpected presence. The Alvey test sentence "*he hears the abbot who did appear to see neither the message with which kim agrees in the abbey nor the abbots whom he agrees with.*" is an example; humans may well have trouble identifying the syntax of this sentence.

7.5 Summary and Conclusion

DIPETT most certainly has limitations, and areas where it can and should be improved. This brief experiment shows that the same can be said for other, better known and more visible parsers. DIPETT appears to perform at least as well as some of the parsers available on the web, and better in some cases. None of the parsers visited produce parse trees that are as detailed as DIPETT's, and DIPETT's parse trees appear to be just as easy (maybe easier) to read. A number of the parsers visited failed to parse simple sentences.

DIPETT should stand up and be counted.

8 Summary, Discussion, and Future Work

This chapter summarizes the contributions of the thesis and presents some ideas for future work in evaluating and optimizing DIPETT.

8.1 Summary

The goal of this thesis was to investigate the use of test suites for evaluating the performance of parsers of English sentences, and to use such a test suite to evaluate DIPETT.

Chapter 2 gives an overview of parser evaluation literature, and establishes that use of a test suite is an appropriate tool and method of diagnostic evaluation.

Chapter 3 describes the design and development of the TSNLP test suite. The TSNLP method was to optimize *progressivity*, *systematicity* and *control* of vocabulary and phenomena interaction in order to construct a test suite that is adequately broad-coverage, focused on specific phenomena, and non-redundant.

Chapter 4 describes the proposal for this thesis: an alternative test suite of test sentences extracted from Quirk *et al.*'s comprehensive English grammar, a standard academic grammar of English. The Quirk test sentences offer several alternatives to the TSNLP suite — guaranteed broad-coverage, not a restricted vocabulary, phenomena-interaction is not controlled. The design of DEVAL, a program for performing progress evaluation is described.

Chapter 5 presents an evaluation of DIPETT v3.0 on the Quirk sentences, and then on the TSNLP sentences. Problems identified in DIPETT v3.0 parse trees for the Quirk sentences are identified, and in some cases possible solutions are discussed. The DIPETT v3.0 parse trees for the TSNLP sentences are then evaluated and a similar problem list is formed. It is determined that the TSNLP test suite is finer-grained and more precise in its diagnostic power, but the Quirk sentences diagnose the same problems and more, although with less specific detail.

Chapter 6 shows how the evaluation results were used to make improvements to DIPETT. The changes in DIPETT's grammar cause changes in its parse tree output. Some new parse trees are generated for sentences that had previously failed to parse, while other parse trees changed. The changes are discussed. Overall, the improvements cause an increase in coverage scores: from 65% to 92% full parses on the TSNLP suite, and from 79% to 85% on the Quirk sentences.

Chapter 7 compares DIPETT's performance to that of other publicly available large-coverage parsers.

Chapter 8 presents a summary, and future work to be done to improve the Quirk test suite and optimize DIPETT.

8.2 Conclusions

The systematic procedure described in this paper helped to improve a rather complex parser of English sentences. Some lessons may be drawn by those who want to use the same method.

8.2.1 Comparison of the Effectiveness of the TSNLP and Quirk Test Items

DIPETT's parse trees for the TSNLP test items were compared with the analysis provided in the TSNLP *tsdb(1)* database. The comparisons are necessarily manual, and therefore not easy. The TSNLP suite is very fine-grained and pinpoints problems very precisely. Its design requires phenomena to be tested in isolation, not in combination. Phenomena coverage of the *tsdb(1)* suite is not properly broad unless it has been extended. The test items for the basic sentence phenomena (complementation, modification, diathesis, modality, tense and aspect, clause type, coordination, and negation) are exhaustively complete.

The Quirk-based test suite complements the TSNLP suite, and is useful in ways other than the TSNLP. The parse analyses of the Quirk sentences are not as readily available as for the TSNLP test items, and the evaluation of the resulting DIPETT parse trees required careful study. The Quirk test sentences are coarser-grained than the TSNLP suite. They are broader

in their phenomena coverage, but not so completely or exhaustively precise in their diagnostic coverage of each particular phenomenon covered. Even though only two chapters of Quirk *et al.* were used in the evaluation, the Quirk sentences broadly diagnosed most of the same problems that were identified by the TSNLP evaluation. The Quirk sentences also identified linguistic phenomena that are not covered by DIPETT and were not covered or diagnosed by the TSNLP sentences either. In particular, less controlled combination of phenomena in the test sentences helped to identify significant problems in DIPETT's grammar.

The TSNLP method is designed to optimize control over test data, progressivity, and systematicity. Their test sentences are deliberately simple and the interaction of phenomena is highly controlled. Our experience shows that a test suite for a truly broad-coverage natural language parser must be corpus-like in its coverage of phenomenon interaction, in that corpora comprise arbitrary (but relevant) combinations of phenomena.

8.2.2 Improvements to DIPETT's Performance

The modifications to DIPETT's grammar were intended to improve DIPETT's performance on the TSNLP suite, and a marked improvement in the number of full parses generated by DIPETT has been achieved — an increase from 65% to 92% full parses. The same improvements also caused a more modest improvement in the coverage of the Quirk sentences, even though no attempt was made to "tune" DIPETT to that test suite. The number of full parses on all of the 578 Quirk sentences rose from 79% to 85%.

I worked with the assumption that a parser produces full parse trees only for correct data. Little improvement was shown in parsing (rejecting) ungrammatical sentences, but no attempt was made to optimize DIPETT for that purpose. Some grammar additions actually permit DIPETT to parse (accept) ungrammatical sentences that were rejected before. This condition is not entirely due to shortcomings in DIPETT's grammar. Before meaningful results for ungrammatical sentences can be achieved, such sentences in the test suites must be pruned, leaving only those found ungrammatical by a *surface* syntactic parser. For such

parsers, the ungrammatical test items for the TSNLP complementation phenomena need not be used; neither should the Quirk sentences that are syntactically correct but semantically deviant.

8.3 Future Work

The test suite derived from Quirk can be improved:

- *Scope*: Extraction of example sentences from two chapters provided interesting results. The test suite can be improved by systematic additions from subsequent Quirk *et al.* chapters.
- *Control and accessibility*: The TSNLP database was designed to be extended. The Quirk test suite would be improved by the development of phenomenon labels and interactions, and systematic classification of the phenomena tested. Combination of the two test suites would provide a very powerful diagnostic tool.

Much interesting work remains to be done in evaluating and improving DIPETT:

- At the very least, DIPETT should parse as much of the TSNLP suite as the TANKA group judges is appropriate. Some of the more British particle constructions may be ignored, but a broad-coverage parser of English such as DIPETT should be able to parse most of the TSNLP test items.
- Solutions have been suggested in chapter 5 for some of the grammar problems identified in DIPETT. Most of those solutions can be implemented with little difficulty, although some experimentation will be necessary to confirm their results.
- An interesting experiment for DIPETT would be to improve its complement assignments by integrating a subcategorization dictionary. Or, such a dictionary could be "learned" by DIPETT, much in the same way that the TANKA semantic module HAIKU learns its case patterns.

- New developments in parser evaluation research should be monitored. It would be interesting to evaluate DIPETT on an annotated corpus, possibly the new grammatical relation scheme of Carroll *et al.* (1998, 1999) should that work progress well.

The evaluation and comparison of the performance of parsing systems is an active research field. Researchers are currently interested in the empirical study of how properties of the parser, grammar, and input affect actual, observed run-time efficiency of parsing systems. Of particular interest are the methods, reference grammars, and test data that will facilitate improved comparability. In particular, a recently announced workshop[27] is intended to help the field focus and converge on a common, pre-standard practice in empirical assessment of parsing systems.

[27] Efficiency in Large-Scale Parsing Systems, a Workshop to be held at the 18th International Conference on Computational Linguistics, Coling 2000

9 References

1. Atwell, E. (1996). Comparative evaluation of grammatical annotation models. In R. Sutcliffe, H. Koch & A. McElligott (Eds.) Industrial Parsing of Software Manuals (25-46). Amsterdam, The Netherlands: Rodopi.

2. Balkan, L., Meijer, S., Arnold, D., Dauphin, E., Estival, D., Falkedal, K., Lehmann, S., & Regnier-Prost, S. (1994a). Test Suite Design Guidelines and Methodology. Report to LRE 62-089 (D-WP2.1). University of Essex, UK.

3. Balkan, L., Meijer, S., Arnold, D., Dauphin, E., Estival, D., Falkedal, K., Lehmann, S., Netter, K., & Regnier-Prost, S. (1994b). Issues in Test Suite Design. Report to LRE 62-089 (D-WP2.1). University of Essex, UK.

4. Balkan, L., Netter, K., Arnold, D., & Meijer, S. (1994c). TSNLP: Test Suites for Natural Language Processing. In Proceedings of the Language Engineering Convention, ELSNET, Centre for Cognitive Science, University of Edinburgh (pp. 17-22). Edinburgh, Scotland.

5. Balkan, L., Frederik, F., & Regnier-Prost, S. (editors) (1996). TSNLP User Manual, Volume 1: Background, Methodology, Customization, and Testing. Technical Report. University of Essex, UK.

6. Barker, K., & Delisle, S. (1996) "Experimental Validation of a Semi-Automatic Text Analyzer". TR-96-01, Department of Computer Science, University of Ottawa.

7. Barker, K., Delisle, S., & Szpakowicz, S. (1998). Test-Driving TANKA: Evaluating a Semi-Automatic System of Text Analysis for Knowledge Acquisition. In *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence (LNAI 1418).* Vancouver, Canada.

8. Black, E., Garside, R. & Leech, G. (Eds.) (1993). Statistically-driven computer grammars of English: The IBM / Lancaster approach. Amsterdam, The Netherlands: Rodopi.

9. Brew, C. & Moens, M. (1998). Data-Intensive Linguistics, HCRC Language Technology Group, The University of Edinburgh. http://www.ltg.ed.ac.uk/~chrisbr/dilbook/dilbook.html.

10. Briscoe, E. & Carroll, J. (1993). Generalised probabilistic LR parsing for unification-based grammars. *Computational Linguistics*, 19(1), 25-60.

11. Briscoe, E. & Carroll, J. (1995). Developing and evaluating a probabilistic LR parser of part-of-speech and punctuation labels. In *Proceeding of the 4th ACL/SIGPARSE International Workshop on Parsing Technologies* (pp. 48-58). Prague, Czech Republic.

12. Carroll, J., Briscoe, E. Calzolari, N. Federici, S. Montemagni, S., Pirrelli, V. Grefenstette, G. Sanfilippo, A. Carroll, G. & Rooth, M. (1997). *SPARKLE WP1 specification of phrasal parsing.* http://www.ilc.pi.cnr.it/sparkle.html.

13. Carroll, J., Briscoe, E. & Sanfilippo, A. (1998). Parser Evaluation: a Survey and a New Proposal. In *Proceedings of the International Conference on Language Resources and Evaluation.* Granada, Spain. ftp://ftp.cogs.susx.ac.uk/pub/users/johnca/lre98-final.ps.

14. Carroll, J., Guido, M. & Briscoe, E. (1999). Corpus Annotation for Parser Evaluation. In *Proceedings of the EACL workshop on Linguistically Interpreted Corpora (LINC).* Bergen, Norway.

15. Cole, R., Mariani, J., Uskoreit, H., Zaenen, A. & Zue, V. (1996). Survey of the state of the art in human language technology. http://www.cse.ogi.edu/CSLU/HLTsurvey/HLTsurvey.html.

16. Delisle, S. (1994). Text Processing without A-Priori Domain Knowledge: Semi-Automatic Linguistic Analysis for Incremental Knowledge Acquisition", Ph.D. thesis, TR-94-02, Department of Computer Science, University of Ottawa.

17. *EAGLES*: Final Report (1995). Evaluation of Natural Language Processing Systems, EAGLES Document EAG-EWG-PR.2. ftp://ftp.ilc.pi.cnr.it/pub/eagles/evaluation/ewg95.ps

18. Estival, D., Falkedal, K., Balkan, L., Dauphin, E., Meijer, S., Netter, K., Oepen, S., & Regnier-Prost, S. (1994a). Analysis of Existing Test Suites. Report to LRE 62-089 (D-WP1). University of Essex, UK.

19. Estival, D., Falkedal, K., Lehmann,S., Balkan, L.,., Meijer, S., Arnold, D., Regnier-Prost, S., Dauphin, E., Netter, K., & Oepen, S. (1994b). Test Suite Design: Annotation Scheme. Report to LRE 62-089 (D-WP2.2). University of Essex, UK.

20. Estival, D., Lehmann, S., Falkedal, K., Compagnion, H., Balkan, L., Fouvry, F. Arnold, D., Klein, J., Baur, J., Netter, K., Oepen, S., Dauphin, E., Regnier-Prost, S., & Lux, V. (1995). The Construction of Test Material. Report to LRE 62-089 (D-WP3.1). ISSCO, Université de Genève.

21. Grover, C., Carroll, J. & Briscoe, E. (1993). *The Alvey Natural Tools Grammar.* 4th release edition.

22. Harrison, P., Abney, S., Fleckenger, D., Gdaniec, C., Grishman, P., Hindle, D., Ingria, B. Marcus, M., Santorini, B., & Strzalkowski, T. (1991). Evaluating syntax performance of parser/grammars on English. In *Proceedings of the Workshop on Evaluating Natural Language Processing Systems, ACL.*

23. Hogenhout, W. & Matsumoto, Y. (1996). Toward a More Careful Evaluation of Broad-Coverage Parsing Systems. In *Proceedings of the International Conference on Computational Linguistics, COLING-96* (pp. 562-567). Copenhagen, Denmark.

24. Lehmann, S., Estival, D., Falkedal, D., Compagnion, H., Balkan, L., Fouvry, F., Baur, J., & Klein, J. (1996a). TSNLP User Manual. Volume 3: Test Data Documentation. Technical report. Istituto Dalle Molle per gli Studii Semantici e Cognitivi (ISSCO). Geneva, Switzerland.

25. Lehmann, S., Oepen, S. Regnier-Prost, S., Netter, K., Lux, V., Klein, J., Falkedal, L., Fouvry, F., Estival, D., Dauphin, E., Compagnion, H., Baur, J., Balkan, L. & Arnold, D. (1996b). TSNLP – test suites for natural language processing. In *Proceedings of the International Conference on Computational Linguistics, COLING-96* (pp. 711-716). Copenhagen, Denmark.

26. Lin, D. (1995). A dependency-based method for evaluating broad-coverage parsers. In Proceedings of the IJCAI-95 (pp. 1420-1425). Montreal, Canada.

27. Marcus, M., Santorini, B. & Marcinkiewicz, M. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics, 19(2), 313-330.

28. Netter, K., Armstrong, S., Kiss, T., Klein, J., Lehmann, S., Milward, D., Regnier-Prost, S. Schäler, R., & Wegst, T. (1998). DiET – Diagnostic and Evaluation Tools for Natural Language Processing Applications. In *Proceedings of the International Conference on Language Resources and Evaluation.* Granada, Spain.

29. Oepen, S., Netter, K., Baur, J., Fettig, T., Klein, J., & Oberhauser, F. (1995). The TSNLP Database – From tsct(1) to tsdb(1). Report to LRE 62-089 (D-WP6.1). Deutsches Forschungszentrum für Künstlich Intelligenz GmbH.

30. Quirk, R. & Greenbaum, S. (1973). *A University Grammar of English*, Longman.

31. Quirk, R., Greenbaum, S., Leech, G., & Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*, Longman.

32. Sanfilippo, A., Barnett, R., Calzolari, N. Flores, S., Hellwig, P., Leech, P., Melero, M., Montemagni, S. Odijk, J., Pirrelli, V., Teufel, S. Villegas, M. & Zaysser, L. (1996). *Subcategorization standards*. Report of the EAGLES Lexicon/Syntax Interest Group. http://www.ilc.pi.cnr.it/

33. Soames, S. & Permutter, D. (1979). *Syntactic Argumentation and the structure of English.* University of California Press, Berkeley. Los Angeles, 1979.

34. Sparck Jones, K. & Galliers, J. (1996). Evaluating Natural Language Processing Systems: An Analysis and Review. Lecture Notes in Artificial Intelligence 1083. Springer-Vorlag. Berlin.

35. Srinivas, B., Doran, C., Hockey, B. & Joshi, A. (1995). Heuristics and parse ranking. In *Proceedings of the 4th ACL/SIGPARSE International Workshop on Parsing Technologies.* Prague, Czech Republic.

36. Srinivas, B., Doran, C., Hockey, B. & Joshi, A. (1996). An approach to robust partial parsing and evaluation metrics. In *Proceeding of the ESSLLI'96 Workshop on Robust Parsing.* Prague, Czech Republic.

37. Srinivas, B., Sarkar, A., Doran, C. & Hockey, B. (1998). Grammar & Parser Evaluation in the XTAG Project. In *Proceedings of the International Conference on Language Resources and Evaluation*. Granada, Spain. http://www.cis.upenn.edu/~anoop/indexpage.html.

38. *TEMAA*: Final Report (1997). TEMAA Deliverable D16. http://www.cst.ku.dk/projects/temaa/D16/d16exp.html

39. Walter, Sharon M. (1998). Book Review. Computational Linguistics, 24(2), 336-338.

40. Winograd, T. (1983). *Language as a Cognitive Process (Volume 1: Syntax)*, Addison-Wesley.

Appendix A: Test Suite Extracted from Quirk *et. al.*

Appendix A shows the test suite that was extracted from Quirk *et. al.* (1985)
The Quirk section number and section heading are shown with a grey background.
The sentences, annotations, and comments regarding their analysis are shown as extracted from Quirk *et. al.*

2.3	**Grammatical Units and Constituents**	*In order to state general rules about the construction of sentences, it is necessary to refer to units smaller than the sentence itself: clauses, phrases, words, morphemes. The relation between one unit and another unit of which it is a part is 'constituency'. Compound sentences imply multiple constituency, i.e., divided into two or more immediate constituents.*
1.	The evenings have turned very cold just recently.	[[The] [evening+s]] [[have] [turn+ed]] [[very] [cold]] [[just] [recent+ly]].
2.	The evenings have turned very cold just recently, but the afternoons have been quite warm.	Compound sentence. [[The] [evening+s]] [[have] [turn+ed]] [[very] [cold]] [[just] [recent+ly]] [but] [[The] [afternoon+s]] [[have] [be+en]] [[quite] [warm]]
2.6	**Chain and Choice relationships**	*Grammatical categories may be identified through relationships of choice or substitution between constituents.*
3.	The weather has been very cold just recently.	{The weather | It} {has been | was} {very cold | cold} {just recently | recently}.
4.	It was cold recently.	Same as #3
5.	The weather has been cold recently.	The weather has been (very) cold (just) recently.
6.	Some students will be working late in their rooms.	[NP [DET:Some] [N:student+s]] [VP [AUX:will] [AUX:be] [V:work+ing]] [ADVP [ADV:late]] [PP [P:in] [NP [DET:their] [N:room+s]]]].
7.	Hurry!	The same item can be describe a sentence, clause, phrase, word, and morpheme
2.8	**Embedding**	*The phenomenon of embedding accounts for the indefinite extensibility of certain units of grammar.*
8.	I have been talking to some students at the college on the other side of the park at the north end.	NP: some students [at [the college [on [the other side [of [the park [at [the north end [of ...
9.	They live on the top floor of a house in the corner of the old square behind the church.	PP: on [the top floor [of [a house [in [the corner [of [the old square [behind [the church...

2.9	**Subordination**	*Clauses which are embedded in other clauses are subordinate clauses and are often introduced by a subordinating conjunction. There are also clauses, especially relative clauses, which are constituents of phrases and are only indirectly embedded within a larger clause.*
10.	The weather has been remarkably warm.	[NP:the weather] [VP:has been] [ADJP: remarkably warm]
11.	We returned from Italy last week.	[VP:returned] [PP:from Italy] [NP:last week]
12.	The weather has been remarkably warm since we returned from Italy last week.	[NP:The weather] [VP:has been] [ADJP:remarkably warm] [ADV CL: [CONJ:since] [NP:we] [VP:returned] [PP:from Italy] [NP:last week]].
13.	The room has a large window which faces south.	[NP:The room] [VP:has] [NP:a large window [RELCL: [PRO:which] [VP:faces] [ADVP:south]]].
14.	This is the malt that lay in the house that Jack built. This is the rat that ate the malt that lay in the house that Jack built.	Embedding gives rise to the theoretical possibility of grammatical units having indefinite length.
2.10	**Coordination**	*Two or more units may constitute a single unit of the same kind. Coordination is typically signalled by a coordinating conjunction.*
15.	It was Christmas Day and the snow lay thick on the ground.	[S-CONJ: [It was Christmas Day] [CONJ: and] [the snow lay thick on the ground]]
16.	You can go by air or by rail.	[PP-CONJ: [PP:by air] [CONJ:or] [PP:by rail]]
17.	His son and daughter live in Buenos Aires.	[NP: [DET: His] [N-CONJ: [N:son] [CONJ: and] [N: daughter]]]
18.	The colours of the rainbow are blue, green, yellow, orange, red, indigo, and violet.	[ADJ-CONJ: [ADJ: blue] [,] [ADJ: green] [,] ... [,] [CONJ: and] [ADJ: violet]]

2.12	**Form and Function**	*A constituent may sometimes vary its position (mobility) and sometimes may be omitted (optionality). An ADVP may be replaced by a different kind of constituent, which is similarly optional and mobile.*
19.	The weather has been very cold just recently	[NP:the weather] [VP:has been] [ADJP:very cold] [ADVP: just recently].
20.	Just recently, the weather has been very cold.	[ADVP: Just recently] [NP:the weather] [VP:has been] [ADJP:very cold].
	An ADVP may be replace by a different kind of constituent, which is similarly optional and mobile.	
21.	The weather has been very cold this month.	[NP:the weather] [VP:has been] [ADJP:very cold] [NP: this month]
22.	The weather has been very cold during the past week.	[NP:the weather] [VP:has been] [ADJP:very cold] [PP: during the past week]
	We cannot always replace a NP by an ADVP or by a PP.	
23.	This month has been very cold.	[NP:this month] [VP:has been] [ADJP:very cold]
24.	* Just recently has been very cold.	
25.	* During the past week has been very cold.	

2.13	**Clause Structure**	*The verb element is most central: its position is normally medial; it is normally obligatory; it cannot normally be moved; it helps determine what other elements may occur. For the opposite reasons, adverbials are the most peripheral elements. Quirk simplified formula: {A} S {A} V {O} {O} {C} {A...)*
26.	Someone was laughing loudly in the next room.	S Someone V was laughing A loudly A in the next room.
27.	My mother usually enjoys parties very much.	S My mother A usually V enjoys O parties A very much.
28.	In 1945 the country became totally independent.	A In 1945 S the country V became C totally independent.
29.	I have been in the garden all the time since lunch.	S I V have been A in the garden A all the time A since lunch.
30.	Mary gave the visitor a glass of milk.	S Mary V gave O the visitor O a glass of milk.
31.	Most people consider these books rather expensive actually.	S Most people V consider O these books C rather expensive A actually.
32.	You must put all the toys upstairs immediately.	S You V must put O all the toys A upstairs A immediately.

2.14	**A 'fixed word-order language'**	*Fixed word-order is not all that fixed. Adverbials are mobile.*
33.	Usually my mother enjoys parties very much.	A Usually S my mother V enjoys O parties A very much.
34.	My mother enjoys parties very much, usually.	S My mother V enjoys O parties A very much, A usually.
	Elements other than adverbials cannot be moved from their S V O sequence.	
35.	* Usually enjoys parties my mother very much.	[A V O S A]
36.	* Enjoys usually my mother parties very much.	[V A S O A]
37.	* My mother parties usually enjoys very much.	[S O A V A]
38.	* Mother my usually enjoys parties much very.	
2.15	**Adverbials**	*Some adverbials are obligatory, others are optional.. Optional adverbials are annotated {A}*
39.	My mother enjoys parties very much.	"usually" is optional
40.	I have been in the garden all the time since lunch.	Quirk et. al. class "in the garden" as an adverbial because it answers the question 'where'. In this case, the adverbial is obligatory.
41.	* I have been all the time since lunch.	"in the garden" is obligatory
42.	You must put all the toys upstairs immediately.	Obligatory adverbial "upstairs"
43.	* You must put all the toys immediately.	"upstairs" is obligatory
44.	To my regret, he refused the offer of help.	Sentence adverbials qualify a whole sentence rather than just part of a clause and are optional.
45.	He was, however, very interested in my other proposals.	[S V {A} C A]

2.16	**Clause types**	*Seven clause types: SV, SV + {O | C | A}, SVO + {O | C | A}* *Three main verb classes: Intransitive (SV), Transitive (SVO, SVOO, SVOC, SVOA), Copula (SVC, SVA)*	
46.	Someone was laughing.	SV	Intransitive verbs are followed by no obligatory element, and occur in type SV.
47.	I have been in the garden.	SVA	Copular verbs are followed by a Subject Complement or an Adverbial and occur in types SVC and SVA.
48.	You must put all the toys upstairs.	SVOA	Transitive verbs are followed by an Object and occur in types SVO, SVOO, SVOC, SVOA.

2.17	**Objects and complements**	*Whenever there are two objects (type SVOO) the former is normally the indirect object, and the latter the direct object. Although the indirect object is more central with respect to position, it is more likely to be optional and may generally be paraphrased by a prepositional phrase functioning as an adverbial.*	
49.	My mother enjoys parties.	SVO	Monotransitive verbs occur in type SVO
50.	Mary gave the visitor a glass of milk.	SVOO	Ditransitive verbs occur in type SVOO
51.	The country became totally independent.	SVC	Copular verbs are followed by a Subject Complement or an Adverbial and occur in types SVC and SVA.
52.	Most people consider these books rather expensive.	SVOC	Complex transitive verbs occur in types SVOC and SVOA
53.	The country became a separate nation.	SVC_s	In SVC clauses the complement applies some attribute to the subject
54.	Most people considered Picasso a genius.	$SVOC_o$	In SVOC clauses the complement applies an attribute to the object.

2.18	**Obligatory adverbials**	*Obligatory adverbials are largely restricted to space adjuncts. Such adjuncts occurring in the SVA and SVOA patterns include not only those indicating position, but also those indicating direction. By extension, they may also include those which specify temporal location. There are some obligatory adverbials to which a locational metaphor cannot be applied.*
55.	He stayed very quiet.	S He $V_{copular}$ stayed C_s very quiet.
56.	He stayed in bed.	S He $V_{copular}$ stayed A_s in bed.
57.	They kept him very quiet.	S They $V_{complex\ transitive}$ kept O him C_o very quiet.
58.	They kept him in bed.	S They $V_{complex\ transitive}$ kept O him A_o in bed.
59.	She put the glass down.	S She V put O the glass A down.
60.	The next meeting will be on the 5th of February.	S The next meeting V will be A on the 5th of February.
61.	The road is under construction.	S The road $V_{copular}$ is A_s under construction.
62.	We kept him off cigarettes.	S We $V_{complex\ transitive}$ kept O him A_o off cigarettes.
63.	They treated her kindly.	S They V treated O her A kindly.
64.	He is without a job.	S He V is A without a job.

2.19	**Clause elements subclassified**	*A_s subject-related adverbial ; C_o object complement ; C_s subject complement ; O_d direct object ; O_i indirect object*
65.	Prices rose.	S Prices $V_{intransitive}$ rose.
66.	Elizabeth enjoys classical music.	S Elizabeth $V_{monotransitive}$ enjoys O_d classical music.
67.	Your face seems familiar.	S Your face $V_{copular}$ seems C_s familiar.
68.	My sister lives next door.	S My sister $V_{copular}$ lives A_s next door.
69.	We all wish you a happy birthday.	S We all $V_{ditransitive}$ wish O_i you O_d a happy birthday.
70.	The president declared the meeting open.	S The president $V_{complex\ transitive}$ declared O_d the meeting C_o open.
71.	The doorman showed the guests into the drawing room.	S The doorman $V_{complex\ transitive}$ showed O_d the guests A_o into the drawing room.

2.21	**Active and passive structures**	*Clauses containing a noun phrase as object are distinguished by the fact that they are usually matched by passive clauses, in which the object noun phrase now appears as subject. In all passive clause types the agent by-phrase, which incorporates a noun phrase equivalent to the subject of the corresponding active clause, has the structural status of an optional adverbial.*
72.	They considered him a genius.	SVOC
73.	He was considered a genius	(by...) S $V_{passive}$ C_s

74.	He seemed a genius.	S V C_s
75.	A number of people saw the accident.	S V O_d
76.	The accident was seen.	S $V_{passive}$
77.	The accident was seen by a number of people.	S $V_{passive}$ A
78.	John had the book.	*Had* cannot be passivized
79.	* The book was had by John.	
80.	My father gave me this watch.	S V O_i O_d
81.	I was given this watch.	S $V_{passive}$ O_d
82.	I was given this watch by my father.	S $V_{passive}$ O_i A
83.	This watch was given to me by my father.	S $V_{passive}$ O_i A
84.	This watch was given me.	S $V_{passive}$ O_i
85.	Queen Victoria considered him a genius.	S V O_d C_o
86.	He was considered a genius.	S $V_{passive}$ C_s
87.	He was considered a genius by Queen Victoria.	S $V_{passive}$ C_s A
88.	An intruder must have placed the ladder there.	S V O_d A_o
89.	The ladder must have been placed there.	S $V_{passive}$ A_s
90.	The ladder must have been placed there by an intruder.	S $V_{passive}$ A_s A

2.22	**Copular and complex transitive structures**	*Another correspondence often obtains between an SVOC clause and a clause with an infinitive or that-clause. The O and C of and SVOC clause are in the same relation to one another as the S and C of an SVC clause.*
91.	I considered her beautiful.	S V O *her* C *adj* beautiful
92.	I considered her to be beautiful.	S V O *her* C *to-inf be* beautiful
93.	I considered that she was beautiful.	S V O *that-cl* [S she V was C *adj* beautiful]

2.23	**Indirect objects and prepositional phrases**	*There is a further correspondence by which SVOO clauses can be converted into SVOA clauses by the substitution of a PP following the Od for the Oi preceding it.*
94.	She sent Jim a card.	S V $O_{indirect}$ O_{direct}
95.	She sent a card to Jim.	S V O A
96.	She left Jim a card.	S V $O_{indirect}$ O_{direct}
97.	She left a card for Jim.	S V O A

2.26	**General structural characteristics of phrases**	*Nonheaded phrases: endocentric; elements are normally obligatory.* *Headed phrases: exocentric; one obligatory element optionally preceded or followed by other elements.*
98.	I went to London.	Prepositional phrase is nonheaded
99.	* I went to.	
100.	* I went London.	
101.	This is very important indeed	Adjective phrase is headed
102.	This is important.	
103.	* This is very indeed.	

2.27	**Verb phrases and noun phrases**	*A nonfinite main verb cannot normally stand on its own in independent clauses; the auxiliaries can, have, and be may not be omitted.*
104.	Jack can play the trombone.	
105.	Our team has been beaten.	
106.	The room contains vases.	
107.	The room contains some beautiful Flemish vases.	
108.	The room contains a vase.	
109.	* The room contains vase.	a singular noun head can no longer stand alone
110.	The room contains a beautiful Flemish vase.	

2.28	**Summary of phrase structures**	*Verb phrases consist of a main verb which either stands alone as the entire verb phrase, or is preceded by up to four verbs in an auxiliary function.*
111.	The ship sank.	S The ship V sank.
112.	The ship was sinking.	S The ship aux was V sinking.
113.	The ship has been sunk.	S The ship aux has been V sunk.
114.	The ship must have been sinking.	S The ship aux must have been V sinking.
115.	The ship may have been being sunk.	S The ship aux may have been being V sunk.
	Noun phrases consist of determinative, premodification, head, postmodification/complementation. The head is typically a noun, and the other elements determine the head, modify the head, or complement another element in the phrase.	
116.	I remember him.	I remember [head] him.
117.	I remember Peter.	I remember [head] Peter.
118.	I remember Alice's wedding.	I remember [det] Alice's [head] wedding.
119.	I remember that girl with the red hair.	I remember [det] that [head] girl [postmod].with the red hair
120.	I remember all those fine warm days in the country last year.	I remember [det] all those [premod] fine warm [head] days [postmod] in the country last year.
121.	I remember a better story than that.	I remember [det] a [premod] better [head] story [compl] than that.
122.	I remember the best trip that I ever had.	I remember [det] the [premod] best [head] trip [compl] that I ever had.
123.	I remember a good trip that I once had.	I remember [det] a [premod] good [head] trip [postmod] that I once had.
	Adjective phrases consist of an adjective as head, optionally preceded and followed by modifying elements	
124.	The weather was pleasant.	The weather was [head] pleasant.
125.	The weather was too hot to be enjoyable.	The weather was [premod] too [head] hot [compl] to be enjoyable.
126.	The weather was incredibly cold.	The weather was [premod] incredibly [head] cold.
127.	The weather was pleasant enough.	The weather was [head] pleasant [postmod] enough.
	Adverb phrases are similar to adjectives, except they have an adverb as their head	
128.	I spoke to him yesterday.	I spoke to him [head] yesterday.
129.	I spoke to him quite often.	I spoke to him [premod] quite [head] often.
130.	I spoke to him severely indeed.	I spoke to him [premod] very [head] severely [postmod] indeed.
131.	I spoke to him as clearly as I could.	I spoke to him [premod] as [head] clearly [compl] as I could.
	Prepositional phrases consist of a preposition followed by a prepositional complement, which is normally a noun phrase.	
132.	I met her for lunch.	I met her [prep] for [prepositional complement] lunch.
133.	I met her at the corner of the street.	I met her [prep] at [prepositional complement] the corner of the street.
134.	I met her on Saturday morning.	I met her [prep] on [prepositional complement] Saturday morning.
135.	I met her by a strange coincidence.	I met her [prep] by [prepositional complement] a strange coincidence.

2.32	**Complementation**	*Complementation is the function of a part of a phrase or clause which follows a word and completes the specification of a meaning relationship which that word implies. Complementation may be either obligatory or optional on the syntactic level and overlaps with other functions, such as adverbials and modifiers.*
136.	He deceived his father.	Obligatory verb complementation
137.	* He deceived.	
138.	He allowed me a respite.	
139.	* He allowed me.	
140.	All sales are subject to tax.	Obligatory adjective complementation
141.	* All sales are subject.	
142.	Mr. Gould is likely to resign.	
143.	* Mr. Gould is likely.	
144.	Joan was eating her lunch.	Optional verb complementation
145.	Joan was eating.	
146.	The boat was ready for departure.	Optional adjective complementation
147.	The boat was ready.	
148.	We always found him cheerful.	Some adjectives may occur as both a modifier and a head
149.	He was a cheerful person.	

2.33	**Modification and complementation**	*Modification always relates to the head of a phrase. The complementing function may relate to a premodifier which is separated from its complementation by the head.*
150.	Greek is a more difficult language than French.	"than French" complements the comparative adverb "more", not the head noun "language"
151.	* Greek is a difficult language than French.	
152.	She was too ill to travel.	"to travel" complements the adverb "too", rather than the head "ill"
153.	* She was ill to travel.	
154.	She is rather young to drive a car.	"rather" modifies "young"
155.	She is young to drive a car.	

2.43	**Word classes in relation to meaning**	*Broadly speaking, nouns can be categorized naturally as stative in that they typically refer to entities that are stable. Verbs can be more naturally characterized as dynamic: they are fitted to indicate action and temporary or changing conditions. Adjectives attribute stable properties and many adjectives contrast with verbs in their inability to be rendered temporary by the progressive aspect*
156.	John works hard.	Dynamic - Adverb
157.	John is working hard.	Dynamic - Adverb
158.	John is tall.	Stative - Adjective
159.	* John is being tall.	
	Adverbs (adjuncts) are dynamic rather than stative in that they add a particular condition of time, place, manner to the dynamic implication of the verb	
160.	Marion is beautiful.	Stative - Adjective
161.	Marion sings beautifully.	Dynamic - Adverb
	Adjectives can take on some of the dynamic implications of the adverb	
162.	John is a hard worker.	Stative - Adjective
163.	Marion is a skilful singer.	Stative - Adjective
164.	* He is knowing English.	Some verbs cannot be used with the progressive aspect and therefore belong to the stative rather than the dynamic category
165.	He is being a nuisance again.	Adjectives are primarily stative although they may resemble verbs in referring to transitory conditions.
166.	He is being naughty again.	

2.44	**Pro-forms**	
167.	John searched the big room and then the small room.	A pronoun can serve as a substitute for a noun
168.	John searched the big room and then the small one.	
169.	The man invited the little Swedish girl because he liked her.	A pronoun tends to be a surrogate for a whole noun phrase
170.	Many students did better than many students expected.	These are not equivalent in meaning
171.	Many students did better than they expected.	
172.	Mary is in London and John is there too.	There are pro-forms for time, place, and other adverbials
173.	Mary arrived on Tuesday and John arrived then too.	
174.	The police searched the big room carefully, but the small room less so.	
175.	He often behaved prudently, but he did not always behave thus.	"Thus" (old English) and "so" may be used as pro-forms for adverbials
176.	He often behaved prudently, but he did not always behave so.	
177.	She hoped that they would clean the house carefully before her arrival, but unfortunately they didn't do so.	"So" substitutes, along with the pro-verb "do" for a main verb and whatever follows in the clause

2.45	**Wh-words**	
178.	Where is Mary?	Wh-words may be regarded as a special set of pro-forms.
179.	Mary is in London.	
180.	John is there too.	
181.	The place where Mary lives is London.	
182.	I wonder where Mary lives.	
183.	Where Mary lives, the traffic is very noisy.	
	Through the use of wh-words we can ask for the identification of the subject, object, complement, or an adverbial of a sentence.	
184.	They make him the chairman every year.	S They V make O him C the chairman A every year.
185.	Who makes him the chairman every year?	wh-word as S
186.	Whom do they make the chairman every year?	wh-word as O
187.	What do they make him every year?	wh-word as C
188.	When do they make him the chairman?	wh-word as A
	Wh- words may also function as determiners, adjectives, and as modifying adverbs	
189.	Which cup is yours?	wh-word as determiner
190.	How do you feel?	wh-word as adjective
191.	How old are you?	wh-word as adverb

2.47	**Subject and predicate**	*Simple sentences are traditionally divided into two major parts, a subject and a predicate. This means that in terms of clause elements, the subject (S) is distinguished from the other elements (V and combinations of O, C, and A) which follow it.*
192.	Julie buys her vegetables in the market.	[subject] Julie [predicate] buys her vegetables in the market.
193.	The train arrived late today.	[subject] The train [predicate] arrived late today.
194.	Tigers are carnivorous.	[subject] Tigers [predicate] are carnivorous.
2.48	**Operator and predication**	*Not all simple statements have an operator, but when it occurs, it is normally the word which directly follows the subject. Provisionally defined as the first or only auxiliary, it has a crucial role in the formation of sentences. By reversing the order of the subject and operator, we can change a statement into a yes-no question. The operator has a similar role in the formation of most wh- questions.*
195.	He had given the girl an apple.	
196.	Had he given the girl an apple?	
197.	John is inviting somebody to dinner.	

198.	Whom is John inviting to dinner?	
199.	The gold has been hidden somewhere.	
200.	Where has the gold been hidden?	
	Negation also makes crucial use of the operator: to make a positive statement negative, we insert not after the operator, or else add to the operator the informal enclitic -n't	
201.	I shall not be working this afternoon.	
202.	We had not given the girl an apple.	

2.49	**DO, BE, and HAVE as operators**	*The definition of the operator as the first auxiliary raises the question of what happens if the corresponding positive declarative has no auxiliary, and therefore no operator. In such cases, in the corresponding interrogative and negative structures, the verb DO is introduced as a "dummy" auxiliary to perform the function of operator.*
203.	They often go abroad.	declarative
204.	Do they often go abroad?	interrogative
205.	Her father works in a factory.	declarative
206.	Where does her father work?	interrogative
207.	We received your letter.	declarative
208.	We did not receive your letter.	negative
	"DO" may function as a main verb	
209.	He did it.	declarative
210.	He did not do it	negative
	Unlike DO, BE functions as an operator even when it constitutes the whole verb phrase, and is thus a main verb.	
211.	Is everything ready?	interrogative
212.	Everything is ready.	declarative
213.	Was Titian a painter?	interrogative
214.	Titian was a painter.	declarative
215.	Are these books for sale?	interrogative
216.	These books are for sale.	declarative
	The main verb HAVE tends to resemble the main verb DO in not functioning as operator although there is a traditional usage in which it does so.	
217.	Do you have a box of matches?	interrogative
218.	Have you a box of matches?	interrogative

2.50	**Questions and negation**	*This section states process rules for forming questions and negative sentences in English, given that we know how to form simple statements.*
219.	Have you borrowed my pencil?	yes-no question
220.	Who has borrowed my pencil?	wh- question
221.	Why have you borrowed my pencil?	wh- question
222.	I haven't borrowed your pencil.	negation
223.	Did you borrow my pencil?	yes-no question with DO as operator
224.	Why did you borrow my pencil?	wh- question with DO as operator
225.	I didn't borrow your pencil.	negation with DO as operator

2.51	**Predications and pro-forms**	*Two predications can be joined by coordination.*
226.	You should eat regularly and take more exercise.	
227.	Someone has broken into the house and stolen the money.	

	English has a pro-form do so which substitutes for a predicate or a predication.	
228.	She hoped that he would search the room carefully and he searched the room carefully.	
229.	She hoped that he would search the room carefully and he did so.	
230.	She hoped that he would search the room carefully but he didn't search the room carefully.	
231.	She hoped that he would search the room carefully but he didn't do so.	
	The predication may be omitted altogether; the operator left stranded by this omission, shows its capability of standing as an independent unit.	
232.	She hoped that he would search the room carefully and he did.	
233.	She hoped that he would search the room carefully but he didn't.	
	The composite pro-form do so should be distinguished from the empty auxiliary DO used in DO-support.	
234.	Yes, they do.	Do they pay you for the work?
235.	No, they don't.	
236.	Yes they will.	Will they pay you for the work?
237.	No, they won't.	
238.	Yes, they are.	Are they paying you for the work?
239.	No, they aren't.	
240.	Yes, they have.	Have they been paying you for the work?
241.	No, they haven't.	

2.52	**Ellipsis**	*Elements of a sentence which are predictable can be omitted.*
242.	Yes, they are paying me for the work.	Unreduced
243.	Yes, they are doing so.	Pro-form
244.	Yes, they are.	Ellipsis

2.54	**Negative forms**	
245.	I saw nobody.	
246.	I didn't see anybody.	
247.	Have you never been to London?	
248.	Haven't you ever been to London?	
249.	John never invites any students to his parties.	
250.	No one ever gives her any encouragement.	

2.55	**Scope**	*SCOPE is the general term that we shall use to describe the semantic 'influence' which such words have on neighbouring parts of a sentence. The position of a negative form is generally significant in defining whatever follows it as nonassertive.*
251.	Some people never send any Christmas cards.	any is within the scope of the negation, some is not
252.	* Any people never send some Christmas cards.	it is not possible to reverse the assertive and nonassertive words
253.	I don't know any people who never send Christmas cards.	any people can follow a negative
	The operator or wh-element which begins a question indicates that what follows is with the scope of interrogation.	
254.	Does he seriously believe that?	
255.	Seriously, does he believe that?	sentence adverbial; not part of the matter being questioned
256.	Why doesn't he also take the children abroad?	

257.	Also, why doesn't he take the children abroad?	

2.57	**Directives and exclamations**	*Directives contain no subject or operator: they consist simply of a predication with an imperative verb, i.e., a verb in its base form: Imperatives can be negative, and for this do is introduced as an imperative marker.*
258.	Be quiet!	Directive
259.	Search the room carefully!	Directive
260.	Make yourself a cup of tea.	Directive
261.	Don't hurry!	Imperative
262.	Don't be frightened!	Imperative
263.	Don't wait for me.	Imperative
264.	What beautiful clothes she wears!	Exclamative
265.	How well Philip plays the piano!	Exclamative

2.59	**Grammatical highlighting**	*The Cleft Sentence enables the user to select which element of the sentence will be highlighted. The cleft sentence is divided into two main parts: an initial focal element, followed by a background structure which resembles a relative clause. Fronting is the initial placing of an element such as an object or adverbial. Extraposition is the postponing of a normally nonfinal element to final position. The introductory word "there" in Existential sentences carries no information in itself, but merely supplies the structural requirement for an initial subject.*
266.	It's Julie who buys her vegetables in the market.	Cleft Sentence
267.	It's her vegetables that Julie buys in the market.	Cleft Sentence
268.	It's in the market that Julie buys her vegetables.	Cleft Sentence
269.	Her vegetables Julie buys in the market.	Fronting
270.	In the market Julie buys her vegetables.	Fronting
271.	What you say doesn't matter.	
272.	It doesn't matter what you say.	
273.	There was someone knocking at the door.	"Someone was knocking at the door."

3.1	**Major verb classes**	*The function of the V element in English clause structure is realized by the Verb Phrase which consists of one or more verb constituents. Verbs can be divided into three major categories according to their function within the verb phrase: Full Verbs (an open class); Primary Verbs (be, have, do); Modal Auxiliary Verbs (such as will, might, etc.)*
274.	She left yesterday.	Full verb *leave*
275.	Has she not left yet?	Primary verb *have* + *leave*
276.	Did she leave yesterday?	Primary verb *do* + *leave*
277.	She won't leave tomorrow.	Modal auxiliary *will* + *leave*
278.	She might be leaving next week.	Modal auxiliary *might* + *leave*

3.2	**Verb forms and the verb phrase**	*The verb forms have different functions in finite and nonfinite verb phrases. On this basis the -s form and the past form are called* Finite *whereas the -ing participle and the -ed participle are called* Nonfinite. *The Base form is sometimes finite, sometimes nonfinite. In a finite verb phrase only the first verb word is finite and subsequent verbs are nonfinite.*
279.	Calling early, she found him at home.	In a nonfinite verb phrase, all verbs are nonfinite
280.	Having been called early, he felt sleepy all day.	
281.	I call regularly.	The Base Form occurs as a finite form in the present tense except -3sg
282.	* He call regularly.	
283.	Call at once.	The imperative
284.	They demanded that she call and see them	The present subjunctive

285.	He may call tonight.	The Base Form occurs as a nonfinite form in the bare infinitive and the to-infinitive
286.	We want her to call.	
287.	He calls every day.	The -S Form occurs as a finite form in 3sg
288.	He's calling her now.	The -ing participle occurs as nonfinite form in the progressive aspect and -ing participle clauses
289.	Calling early, I found her at home.	
290.	Someone called yesterday.	The Past Form occurs as a finite form in the past tense
291.	He has called twice today.	The -ed participle occurs as a nonfinite form in the perfective aspect and the passive voice following BE
292.	Her brother is called John	
293.	Called early, he ate a quick breakfast.	

3.21	**Verbs in auxiliary function**	*The Primary verbs Be, Have, and Do and the Modal verbs can, may ,will, shall, could, might, would, should, and must are capable of functioning as Auxiliary verbs. Auxiliaries have one important syntactic function in common, their ability to act as Operator when they occur as the first verb of a finite verb phrase; as such they are used in the formation of yes-no questions.*
294.	Is he asking any questions?	He is asking (any) questions.
295.	Has he been asking any questions?	He has been asking (any) questions.
296.	Was he asked any questions?	He was asked (any) questions.
297.	Will he be asked any questions?	He will be asked (any) questions.
298.	Has he asked any questions?	He has asked (any) questions.
299.	Does he ask any questions?	He does ask (any) questions.
	Be and have also have this function as main verbs	
300.	Is she a tall girl?	She is a tall girl.
301.	Has he any money?	He has (any) money.
302.	He might have been being questioned by the police.	A complex verb phrase

3.22	**Operator in negation with not**	*In forming negative finite clauses, the first auxiliary is placed before the negative word not. Full verbs are distinguished from auxiliary verbs by their inability to form negation in this way.*
303.	* She saw not the play.	
304.	* She not can do it.	
3.23	**Negative and verb contractions**	*The negative word not following an operator can in most cases be contracted and attached as an enclitic particle to the auxiliary. In addition, many operators have contracted nonnegative forms. Nonnegative contractions cannot combine with negative contractions to form doubly-contracted forms. Contracted forms, being enclitic to a preceding word, do not occur where the operator is the only verb in the verb phrase and precedes an ellipsis.*
305.	She isn't studying.	
306.	She's not studying.	
307.	* She'sn't studying.	
308.	* 'll you be in tonight?	Will you be in tonight?
309.	* No, but I'll tomorrow night.	No, but I will tomorrow night.
3.24	**Inversion of subject and operator**	*Auxiliaries admit inversion especially in interrogative clauses.*
310.	Will she come?	
311.	* Plans she to come?	
312.	Does she plan to come?	main verbs require the use of DO
313.	At no time was the entrance left unguarded.	Inversion of subject and operator occurs not only in interrogatives but also in sentences with introductory negatives or semi-negatives.

3.26	**Operator in reduced clauses**	*Auxiliaries can function as operators in a range of such reduced constructions where the verb is omitted either by ellipsis or by proform substitution.*
314.	Ann will stay and so will Barbara.	So / neither / nor + Operator
315.	Bill stayed and so did Henry.	
316.	Ann won't stay and neither will Barbara.	
317.	Bill didn't stay, nor did Henry.	
318.	Ann will stay late and Barbara will too.	Operator + Too / either
319.	Bill broke his promise, and Henry did too.	
320.	Ann won't eat much and Barbara won't either	
321.	Bill didn't break his promise, and Henry didn't either.	
322.	Ann said she would be late, and late she was.	Predication Fronting
323.	Bill said he would win the match, and win the match he did.	
324.	Ann said she would be late, which she was.	Relativized Predication
325.	Bill said he would win the match, which he did.	
326.	Ann hoped that we would stay, but unfortunately we couldn't.	There are also reduced constructions in which the two auxiliaries may differ
327.	You should take a break whenever you can.	

3.27	**Pre-adverb position**	*Frequency subjuncts (e.g., always, never) and disjuncts (e.g., certainly, probably) typically (not necessarily) follow auxiliaries as operators whereas they precede the main verb.*
328.	She never believed his story.	
329.	She probably believed his story.	
330.	She would never believe that story.	
331.	She would probably believe that story.	
332.	She would probably never have believed his story.	If there is more than one auxiliary, the adverb will still generally occur after the first, i.e., the operator.
333.	She probably never would have believed that story.	
334.	She would probably have never believed that story.	
335.	* She believed never his story.	Such adverbs could not occur immediately after the main verb except where the main verb is BE (and is therefore an operator).
336.	* She believed probably his story.	
337.	She was probably a taxpayer.	

3.28	**Quantifier position**	*Quantifiers which modify the subject of the clause may occur after the operator as an alternative to the predeterminer position*
338.	All the boys will be there.	
339.	The boys will all be there.	
340.	Both my parents are working.	
341.	My parents are both working.	
342.	All our team played well.	quantifiers *all* and *each* do not occur after a main verb
343.	* Our team played all well.	
344.	Each of us owns a bicycle.	
345.	* We own each a bicycle.	
346.	Our team all played well.	The quantifier may be placed before the main verb.
347.	We each own a bicycle.	

3.29	**Independence of subject**	*Auxiliaries are semantically independent of the subject.*
348.	The man ought to be here at five.	
349.	The man hopes to be here at five.	
350.	The bus ought to be here at five.	
351.	* The bus hopes to be here at five.	
352.	There used to be a school on the island.	
353.	* There hoped to be a school on the island.	
354.	Thousands of people will meet the president.	Is equivalent to
355.	The president will be met by thousands of people.	
356.	Thousands of people hope to meet the president	Not equivalent to
357.	The president hopes to be met by thousands of people.	

3.30	**Additional features of modal auxiliary verbs**	*Finite functions only: Modal auxiliaries can only occur as the first (operator) element of the verb phrase. They cannot occur in nonfinite functions.*
358.	You will be asked questions.	Construction with the bare infinitive: Modal auxiliaries are normally followed by the infinitive, which is bare, except with *used* and *ought*.
359.	You ought to comb your hair.	
360.	They might have stolen it.	
361.	He used to read for hours.	
362.	You must write.	No third person inflection: Modal auxiliaries have no -s form
363.	She must write.	
364.	* She must writes.	
365.	* She like to write.	
366.	I think he may retire next May.	Abnormal time reference: past forms of the modal auxiliaries can be used to refer to the present and future.
367.	I think he might retire next May.	
368.	Will you phone him tomorrow?	
369.	Would you phone him tomorrow?	
370.	I told him he must be home early.	Modal auxiliaries which do not have a distinct past form can be used to refer to the past in indirect speech.
371.	I told him he had to be home early.	

3.31	**The primary verbs BE, HAVE, and DO**	*Primary verbs as auxiliaries share an association with the basic grammatical verb categories of tense, aspect, and voice. They are distinguished from the modal verbs which are associated with the expression of modal meanings such as possibility, obligation, and volition.*
372.	Ann is a happy girl.	BE as a main verb with copular function:
373.	Is that building a hotel?	
374.	Ann is learning Spanish.	BE as an aspect auxiliary
375.	The weather has been improving.	
376.	Ann was awarded a prize.	BE as a passive auxiliary
377.	Our team has never been beaten.	

3.33	**HAVE**	*Have functions both as an auxiliary and as a main verb. As an auxiliary for perfective aspect, Have combines with an -ed participle to form complex verb phrases*
378.	I have finished.	an auxiliary for perfective aspect
379.	I've not seen her.	
380.	What has she bought?	
381.	They may have been eaten.	

382.	I have no money.	As a main verb, Have normally takes a direct object:
383.	Have you had lunch?	As -ed participle, had is restricted to use as a main verb or to use in the Have to construction.
384.	They have had to sell their car.	
385.	We don't have any money.	When used as a main verb with stative meaning Have combines with DO-support in forming constructions with an operator
386.	Do you have a lighter?	
387.	We haven't any money.	But also acts as an operator itself
388.	Have you a lighter?	
389.	John has got courage.	The informal Have got construction is perfective in form, nonperfective in meaning and is frequently preferred as an alternative to stative Have
390.	* John does have got courage.	
391.	We haven't got any butter.	Three alternatives to express stative senses.
392.	We don't have any butter.	
393.	We haven't any butter.	

3.37	DO	DO as a main verb has the full range of forms. The term DO-support applies to the use of DO as an empty operator (2.49) in conditions where the construction requires an operator but there is no semantic reason for any other operator to be present. All uses of DO as an auxiliary com under this heading.
394.	She doesn't want to stay.	Indicative clauses negated by not where the verb is simple present or simple past
395.	I didn't like mathematics at school.	
396.	Never did he think the book would be finished so soon.	Also includes inversion after an initial negative element.
397.	They do want you to come.	In emphatic constructions where the verb is simple present of simple past and also tag questions where the dummy operator is not accompanied by a main verb
398.	Michael did say he would be here at nine, didn't he?	
399.	Do be quiet.	Persuasive imperative introduced by do.
400.	I haven't done much, I'm afraid.	DO as main verb
401.	I have been meaning to mend that radio but I haven't done it yet.	DO can combine with a pronoun object to act as a pro-predication (referring to some unspecified action).
402.	Why are you doing that?	
403.	She didn't earn so much as she might have done.	DO is used intransitively as a pro-predication.

3.39	**Modal auxiliaries**	
404.	He cannot go.	Operator in negation
405.	He hopes not to go.	this is only acceptable in the case of the negation of to go
406.	He can't go.	Negative contraction
407.	* He hopen't go.	
408.	Can we go?	Operator in inversion
409.	* Hope we to go?	
410.	Yes, I do hope to come.	Emphatic positive
411.	* Yes, I do can come.	
412.	I can come if you can.	Operator in reduced clause
413.	We can always go early.	Position of adverb
414.	We always hope to go early.	
415.	They can all come.	Postposition of quantifier
416.	They all hope to come.	
417.	They all can come.	
418.	It can be done by Ann.	Independence of subject

419.	* It hopes to be done by him.	
420.	I can go.	Bare infinitive
421.	* I hope go.	
422.	I want to go.	No nonfinite forms
423.	* I want to can.	
424.	* I am canning to go.	
425.	I am hoping to go.	
426.	* I had canned to go.	
427.	I had hoped to go.	
428.	She hopes to come.	No "-s" form
429.	* She cans come.	

3.42	**dare and need**	
430.	He needed to escape.	Positive main verb
431.	He needn't escape.	negative modal auxiliary
432.	He doesn't need to escape.	negative main verb
433.	Need we escape?	interrogative modal auxiliary
434.	Do we need to escape?	interrogative main verb
435.	Needn't he escape after all?	negative-interrogative modal auxiliary
436.	Dare he not escape?	
437.	Doesn't he need to escape after all?	negative-interrogative main verb
438.	Doesn't he dare to escape?	
439.	The king was so hot-tempered that no one dare tell him what to do.	As a modal dare exhibits abnormal time reference in that it can be used for past as well as present time.

3.43	**Ought to**	
440.	You ought to stop smoking.	
441.	You oughtn't to smoke so much.	
442.	Ought you to smoke so much?	
443.	Yes, I think I ought to.	
444.	Yes, I think I ought.	

3.43	**Used to**	
445.	She used to attend regularly.	...was in the habit of
446.	I used to be interested in bird-watching.	...was formerly
447.	He usen't to smoke.	British
448.	He used not to smoke.	
449.	He didn't use to smoke.	British and American
450.	He didn't used to smoke.	
451.	Did he use to smoke?	
452.	He used to smoke, didn't he?	

3.45	**Modal idioms**	had better, would rather, HAVE got to, BE to	
453.	We had better leave soon.		
454.	We'd better leave soon.		
455.	I'd rather not say anything.		
456.	They've got to leave immediately.		
457.	The conference is to take place in Athens.		
458.	* I will have got to leave soon.		None of these idiomatic verbs has nonfinite forms, they cannot therefore follow other verbs in the verb phrase. In this sense they are not like main verbs.
459.	* The conference has been to take place in Athens.		
460.	Hadn't we better lock the door?		They are also not entirely like auxiliaries since they do not behave as operators.
461.	Would you rather eat in a hotel?		
462.	We haven't got to pay already, have we?		
463.	I wasn't to know that you were waiting.		
464.	I'd rather not stay here all alone.		Negation
465.	You'd better not lock the door.		
466.	* I'd not rather leave.		The contracted verb form followed by not is unacceptable
467.	* You'd not better resign.		
468.	I'd rather rent the cottage.		Would rather is incapable of active-passive synonymy
469.	* The cottage would rather be rented by me.		
470.	She has got to leave by tomorrow.		HAVE GOT TO and BE TO are like main verbs in that they have an -s form and normal present / past tense contrast

3.47	**Semi-auxiliaries: "be going to" etc.**	*The semi-auxiliaries consist of a set of verb idioms which express modal or aspectual meaning and which are introduced by one of the primary verbs HAVE and BE: e.g., have to, or be X to where X is one of: able, bound, likely, supposed, about, due, meant, willing, apt, obliged.*	
471.	Ada isn't going to win.		
472.	* Ada doesn't be going to win.		
473.	Is Ada going to win?		
474.	* Does Ada be going to win?		
475.	* He is goingn't to. * He is going ton't.		To be comparable to an auxiliary in its entirety, be going to would have to form its negation by adding not to its second or third word:
476.	Brazil is going to win the World Cup.		They do resemble auxiliaries in permitting synonymous passives and there-constructions.
477.	The World Cup is going to be won by Brazil.		
478.	Several home teams are going to be beaten tomorrow.		
479.	There are going to be several home teams beaten tomorrow.		
480.	* Several home teams go to be beaten tomorrow.		There is no progressive construction for *be going to*
481.	He was bound to be a failure.		There is no active equivalent of *be bound to*
482.	Someone bound him to be a failure.		
483.	Someone is going to have to complain.		These constructions can occur in combination with preceding auxiliaries.
484.	No one is likely to be able to recognize her.		
485.	We haven't been able to solve the problem.		Nonfinite semi-auxiliaries can fill slots where modal auxiliaries of equivalent meaning cannot occur.
486.	* We haven't could solve the problem.		
487.	To be allowed to speak freely is a human right.		
488.	* To can speak freely is a human right.		

3.48	have to	Have to is the only semi-auxiliary beginning with HAVE rather than BE. Have to differs from have got to by its occurrence in the full range of nonfinite forms.
489.	I may have to leave early.	modal
490.	* I may have got to leave early.	
491.	People are having to boil their drinking water during this emergency.	
492.	The administration has had to make unpopular decisions.	
493.	These days you must work hard if you want to succeed.	Have to can stand in for must in past constructions where must cannot occur.
494.	In those days you had to work hard if you wanted to succeed.	
495.	There must be some solution to the problem.	
496.	There had to be some solution to the problem.	
497.	Do we have to get up early tomorrow.	Have to patterns either as a main verb or as any auxiliary with respect to operator constructions.
498.	Have we to get up early tomorrow?	

3.49	Catenatives	*The term Catenative is used to denote verbs such as "appear to", "come to", "get to" followed by the infinitive.*
499.	Sam appeared to realize the importance of the problem.	"Appear to"
500.	Sam came to realize the importance of the problem.	"Come to"
501.	Sam failed to realize the importance of the problem.	"Failed to"
502.	? Sam got to realize the importance of the problem.	
503.	Sam seemed to realize the importance of the problem.	"Seemed to"
504.	Sam didn't appear to realize the importance of the problem.	Such constructions have meanings related to aspect or modality but are nearer to main verb constructions than semi-auxiliaries, patterning like main verbs in taking DO-support.
505.	Sam didn't come to realize the importance of the problem.	
506.	The importance of the problem appeared to be realized by Sam.	Most of them resemble auxiliary constructions in satisfying the 'independence of subject' criterion Thus the corresponding passive.
507.	The importance of the problem came to be realized by Sam.	
508.	? The importance of the problem failed to be realized by Sam.	
509.	* The importance of the problem got to be realized by Sam.	
510.	The importance of the problem seemed to be realized by Sam.	
511.	John attempted to attack the burglar.	Catenative constructions are not syntactically related to transitive verb constructions in which the verb is followed by a direct object or prepositional object.
512.	John attempted an attack on the burglar.	
513.	John appeared to attack the burglar.	
514.	* John appeared an attack on the burglar.	
515.	The girl started out working.	Other catenative verbs include BE combining with the -ing participle in progressive constructions or with the -ed participle is passive constructions.
516.	The girl kept working.	
517.	The girl kept on working.	
518.	The girl went on working.	
519.	Our team got beaten by the visitors.	

3.53	Nonfinite verb phrases	*The infinitive, the -ing participle and the -ed participle are the nonfinite forms of the verb. Any phrase in which one of these verb forms is the first or only word is a nonfinite verb phrase. Such phrases do not normally occur as the verb phrase of an independent clause.*
520.	He smokes.	finite
521.	To smoke like that must be dangerous.	nonfinite
522.	Mary is having a smoke.	finite
523.	I regret having started to smoke.	nonfinite

524.	He must smoke forty a day.	finite
525.	The cigars smoked here tend to be expensive.	nonfinite
526.	You have been smoking all day.	finite
527.	That was the last cigarette to have been smoked by me.	nonfinite

3.54	**Simple and complex verb phrases**	*The finite verb phrase is "simple" when it consists of only one word, which may be present, past, imperative, or subjunctive. The verb phrase is complex when it consists of two or more words. There are four basic types of construction in a complex verb phrase. These four basic constructions also enter into combinations with each other.*	
528.	Work harder!	Imperative	*Type A (Modal) consists of a modal auxiliary + the base of a verb*
529.	It is important that he work hard.	Subjunctive	*Type B (Perfective) consists of the auxiliary Have + the -ed participle of a verb*
530.	He may have examined it.	AB — Modal + perfective	*Type C (Progressive) consists of the auxiliary BE + the -ing participle of a verb*
531.	He may be examining it.	AC — Modal + progressive	*Type D (Passive) consists of the auxiliary BE + the -ed participle of a verb*
532.	He may be examined.	AD — Modal + passive	
533.	He has been examining it.	BC — Perfective + progressive	
534.	He has been examined.	BD — Perfective + passive	
535.	He is being examined.	CD — Progressive + passive	
536.	He may have been examining it.	ABC — Modal + perfective + progressive	
537.	He may have been examined.	ABD — Modal + perfective + passive	
538.	He may be being examined.	ACD — Modal + progressive	
539.	He has been being examined.	BCD — Perfective + progressive + passive	
540.	He may have been being examined.	ABCD — Modal + perfective + progressive + passive	

3.56	**Simple and complex nonfinite verb phrases**	*Unlike finite verb phrases, nonfinite verb phrases have no tense or mood distinctions and cannot occur in construction with a subject of a main clause. Since modal auxiliaries have no nonfinite forms, they cannot occur in nonfinite verb phrases. There are two types of nonfinite -ing participle phrases. One is progressive, the other is not.*
541.	Smoking cigarettes is dangerous.	not progressive
542.	They caught him smoking cigarettes.	Progressive, while he was smoking cigarettes
543.	We're glad to have invited you.	Infinitives
544.	I'd like to be working.	
545.	I'd hate to be questioned about it.	
546.	I'm glad to have been working.	
547.	He's said to have been invited.	
548.	I expected to be being interviewed then.	
549.	Having invited you, I expected you to come.	Participles
550.	You can probably get an extension on the grounds of being teaching.	
551.	When questioned, he denied everything.	
552.	Having been working all day, I'm very tired.	
553.	Having been invited, he should have come.	
554.	I saw her being interviewed.	

3.57	**Gradience between one and two verb phrases**	*If a nonfinite verb phrase follows a finite one, it is possible for the same construction to be repeated in each phrase.*
555.	We had hoped to have finished by then.	perfective + perfective
556.	I am hoping to be seeing her tomorrow.	progressive + progressive

557.	? They must have been expecting to have been being paid.	More complex possibilities rarely occur.
558.	Sarah and I are going to be leaving tonight.	With semi auxiliaries it is possible to repeat the same construction.
559.	The walls were supposed to be repainted.	
560.	If you'd done that, we would have had to have arrested you.	If you'd done that, we would have had to arrest you.

3.59	**The mandative subjunctive**	*This most common use of the subjunctive occurs in subordinate that- clauses and consists of the base form of the verb only. The that-clause must be introduced by an expression of demand, recommendation, etc. This expression may be in the form of a verb, adjective, or noun. The absence of DO-support and an -s inflection is a criterion distinguishing the present subjunctive from the present indicative.*
561.	The committee proposes Mr. Day be elected.	
562.	The committee proposes that Mr. Day be elected.	no backshifting of tense -- present and past variants are indistinguishable
563.	The committee proposed that Mr. Day be elected	
564.	I demand that the committee reconsider its decision.	
565.	His sole requirement was that the system work.	
566.	They recommend that this tax be abolished.	verb
567.	It is appropriate that this tax be abolished.	adjective
568.	We were faced with the demand that this tax be abolished.	noun
569.	They insisted that we not eat meat.	subjunctive
570.	They insisted that we do not eat meat.	indicative
3.61	**The present subjunctive**	
571.	Even if that be the official view, it cannot be accepted.	Clauses of condition and concession
572.	If that be the official view, it cannot be accepted.	
573.	The President must reject this proposal, lest it cause strife and violence.	Clauses of condition or negative purpose introduced by lest or for fear that.
3.62	**The were-subjunctive**	*The were-subjunctive or past subjunctive is used in adverbial clauses introduced by such conjunctions as if, as if, though, as though etc. and in nominal clauses after verbs like wish and suppose. This subjunctive is limited to the one form, were.*
574.	If I were rich, I would buy you anything you wanted.	
575.	If I was rich, I would buy you anything you wanted.	
576.	Tim always speaks quietly on the phone, as_though he were telling a secret.	
577.	Tim always speaks quietly on the phone, as_though he was telling a secret.	
578.	I wish the journey were over.	

3.64	**Voice**	*Voice is a grammatical category which makes it possible to view the action of a sentence in either of two ways, without change in the facts reported. The passive adds a form of the auxiliary BE followed by -ed participle of the main verb.*
579.	She misses it.	*Active forms:* Present
580.	She missed it.	Past
581.	She may miss it.	Modal
582.	She has missed it.	Perfective
583.	She is missing it.	Progressive
584.	She may have missed it.	modal + perfective
585.	She may be missing it.	modal + progressive
586.	She has been missing it.	perfective + progressive
587.	She may have been missing it.	modal + perfective + progressive
588.	She is missed.	*Passive Forms:* Present
589.	She was missed.	Past

590.	She may be missed.	Modal
591.	She has been missed.	Perfective
592.	She is being missed.	Progressive
593.	She may have been missed.	modal + perfective
594.	She may be being missed.	modal + progressive
595.	She has been being missed.	perfective + progressive
596.	She may have been being missed.	modal + perfective + progressive

3.66	**The passive auxiliaries: be and get**	*The passive auxiliary is normally be. The verb get, which is not normally considered to be an auxiliary, may be used but usually in constructions without an expressed animate agent.*
597.	The cat got run over.	
598.	James got beaten last night.	
599.	James got caught by the police.	Get with an animate agent is possible.
600.	We are getting bogged down in all sorts of problems.	Get is more common as a 'resulting copula' nd may be best analyzed as such in sentences which look superficially like passives, but which could not be expanded by an agent
601.	I have to get dressed before eight o'clock.	
602.	I don't want to get mixed up with the police again.	
603.	You argument gets a bit confused here.	

3.68	**Verb constraints**	*In addition to copular and intransitive verbs, some transitive verbs called 'middle' verbs do not occur in some senses in the passive. All these belong to the class of 'being' and 'having'*
604.	The coat does not fit you.	ACTIVE ONLY
605.	* You are not fitted by the coat.	
606.	The police want him.	Other stative verbs, such as those of volition or attitude, can occur in the passive.
607.	He is wanted by the police.	
608.	Betty was said to be a good teacher.	PASSIVE ONLY
609.	* They said her to be a good teacher.	
610.	He was born in Tubingen.	
611.	? His mother bore him in Tubingen.	
612.	The wanted man fell into the water and was drowned.	does not mean that someone drowned him, i.e., that an agent is implied

3.69	**Prepositional verbs**	*Prepositional verbs are verbal idioms consisting of a lexical verb followed by a prepositions. These can often occur in the passive, but not so freely as in the active.*
613.	The engineers went very carefully into the problem.	
614.	They eventually arrived at the expected result.	
615.	The engineers went very carefully into the tunnel.	
616.	They eventually arrived at the splendid stadium.	
617.	The problem was very carefully gone into by the engineers.	
618.	The expected result was eventually arrived at.	

3.70	**Object constraints**	*Transitive verbs can be followed either by phrasal or by clausal objects. With clauses as objects, the passive transformation is restricted in use.*
619.	John thought that she was attractive.	Finite clause as object.
620.	? That she was attractive was thought by John.	
621.	John hoped to meet her.	Nonfinite clause, infinitive

622.	? To meet her was hoped by John.	
623.	John enjoyed seeing her.	Nonfinite clause, participle
624.	? Seeing her was enjoyed by John.	

3.74	**The passive gradient**	*The purely formal definition of the passive, that the clause contains the construction be (or get) + -ed participle would include all the following sentences. However, not all are considered to be passive.*
	Central passives have a direct active-passive relation.	
625.	This violin was made by my father.	My father made the violin.
626.	This conclusion is hardly justified by the results..	The results hardly justify the conclusion
627.	Coal has been replaced by oil.	? Oil has replaced coal. OR ? (people in many countries) have replaced coal by oil.
628.	This difficulty can be avoided in several ways	<The agent> can avoid the difficulty in several ways.
	Semi-passives represent a mixed or semi-passive class whose members have both verbal and adjectival properties.	
629.	We are encouraged to go on with the project.	(very) enouraged
630.	Leonard was interested in linguistics.	(very) interested
	Psuedo-passives have neither an active transform nor a possibility of agent addition	
631.	The building is already demolished.	
632.	The modern world is getting more highly industrialized and mechanized.	
	The following can be clearly analyzed as having an adjectival complement following a copular verb, the possibility of inserting very confirms the adjectival status of tired.	
633.	My uncle was very tired.	
634.	My uncle got tired.	
635.	My uncle seemed tired.	

5.1	**Types of noun phrase**	
636.	The girl is Mary Smith.	
637.	The blonde girl is Mary Smith.	Premodifiying adjective
638.	The blonde girl in blue jeans is Mary Smith.	Postmodifying prepositional phrase
639.	The blonde girl wearing blue jeans is Mary Smith.	Postmodifying nonfinite clause
640.	The blonde girl who is wearing blue jeans is Mary Smith.	Postmodifying relative clause
641.	She is Mary Smith.	Personal pronoun
642.	* The blonde she is Mary Smith.	Personal pronouns cannot occur with premodification
643.	? She in blue jeans is Mary Smith	Personal pronouns normally do not occur with postmodification

5.2	**Noun classes: count, non-count, and proper nouns**	
	Proper nouns	
644.	I saw Matthew.	
645.	* I saw the Matthew.	
646.	* I saw a Matthew.	
647.	* I saw some Matthew.	"some" is entered into DIPETT's dictionary as a pronoun, a determiner, and as an adverb
648.	* I saw Matthews.	
	Count nouns: individual countable entities	
649.	* I saw book.	This sentence is ungrammatical and should fail
650.	I saw the book.	
651.	I saw a book.	

652.	* I saw some book.	As a determiner, "some" may occur with singular count nouns, meaning "certain"
653.	I saw books.	
	Mass nouns: non-count nouns	
654.	I saw furniture.	
655.	I saw the furniture.	
656.	* I saw a furniture.	
657.	I saw some furniture.	
658.	* I saw furnitures.	
	Nouns that are both count and non-count	
659.	I saw brick.	
660.	I saw the brick.	
661.	I saw a brick.	
662.	I saw some brick.	
663.	I saw bricks.	

Appendix B: DIPETT v3.0 Performance on the Quirk Sentences

Appendix B shows the performance of DIPETT v3.0 on the test suite that was extracted from Quirk.
The Quirk section number and section heading are shown with a gray background.
The sentences, an extraction of DIPETT's parse tree, and evaluation comments are tabulated.

- ✓✓ Indicates that a full parse tree was produced and the representation is correct.
- ✓✗ Indicates that a parse tree was produced but the representation is incorrect.
- ✗ Indicates that no full parse tree was produced for a grammatically correct sentence.
- ✈ Indicates that a full parse tree was generated for a grammatically incorrect sentence.

- S Subject
- V Verb
- O Object
- C Complement
- A Adverb
- {A} Subordinate adverbial, initial or final unless specifically marked medial

2.3 - 2.8	*Grammatical Units and Constituents ; Chain and Choice relationships ; Embedding*		
1. ✓✗	The evenings have turned very cold just recently.	S the evenings $V_{copular}$ have turned C/A *conj-adjs-implicit-and* very cold just recently	"just recently" should be an adverb phrase
2. ✓✗	The evenings have turned very cold just recently, but the afternoons have been quite warm.	S the evenings $V_{copular}$ have turned C/A *conj-adjs-implicit-and* very cold just recently {A} *subord* [but S the afternoons $V_{copular}$ have been C/A *conj-adjs-implicit-and* quite warm]	"just recently" should be an adverb phrase and "quite" is not interpreted as a modifier of "warm"
3. ✓✗	The weather has been very cold just recently.	S the weather $V_{copular}$ has been C/A *conj-adjs-implicit-and* very cold just recently	"just recently" should be an adverb phrase
4. ✓✗	It was cold recently.	S it $V_{copular}$ was C/A *conj-adjs-implicit-and* cold recently	"recently" should be an adverb, not a conjoined adjective
5. ✓✗	The weather has been cold recently.	S the weather $V_{copular}$ has been C/A *conj-adjs-implicit-and* cold recently	"recently" should be an adverb, not a conjoined adjective
6. ✓✗	Some students will be working late in their rooms.	S some students V_{active} will be working O *PP* [P late in] [*NP* their rooms]	"late" interpreted as part of the prepostion, not an adverb
7. ✓✓	Hurry!	$V_{imperative}$ hurry	
8. ✓✓	I have been talking to some students at the college on the other side of the park at the north end.	S ¦ V_{active} have been talking O *conj-pps-implicit-and* to some students at the college on the other side *postmodif* [*conj-pps-implicit-and* of the park at the north end]	
9. ✗	They live on the top floor of a house in the corner of the old square behind the church.	Timed out after 5 minutes	

2.9 – 2.10	*Subordination ; Coordination*		
10. ✓✗	The weather has been remarkably warm.	S the weather $V_{copular}$ has been C/A *conj-adjs-implicit-and* remarkably warm	"remarkably" not interpreted as modifier of "warm"
11. ✓✗	We returned from Italy last week.	S we V_{active} returned O *PP* from *head-noun* week *premodif* Italy last	"last week" not interpreted as an adverb
12. ✓✗	The weather has been remarkably warm since we returned from Italy last week.	S the weather $V_{copular}$ has been C/A *conj-adjs-implicit-and* remarkably warm {A} *subord* since [S we V_{active} returned O *PP* from *head-noun* week *premodif* Italy last]	"last week" not interpreted as an adverb

13. ✓✗	The room has a large window which faces south.	S the room V has O a large window *postmodif [rel-cl-which* S a large window V_{active} faces] {A} south	"south" modifies the main verb "have", not "face"
14. ✓✓	This is the rat that ate the malt that lay in the house that Jack built.	S this $V_{copular}$ is C/A the rat *postmodif [rel-cl-that* S the rat V ate O the malt *postmodif [rel-cl-that* S the malt V lay O *PP* in the house *postmodif [rel-cl-that* S Jack V build O the house]]]	
15. ✓✗	It was Christmas Day and the snow lay thick on the ground.	S it $V_{copular}$ was C Christmas *postmodif [rel-cl-statement* S Day and the snow V lay O Christmas O/C/A *PP* thick on the ground]	Coordinator "and" interpreted as coordinating noun phrases rather than sentences. "thick" not interpreted as an adverb.
16. ✓✗	You can go by air or by rail.	S you V can go O *conj-PP-or* by air by rail	Prepositional phrase is an object rather than an adverbial
17. ✓✗	His son and daughter live in Buenos Aires.	S his *conj-nps-and* son daughter V live O *PP* in Buenos Aires	Prepositional phrase is an object rather than an adverbial
18. ✓✓	The colours of the rainbow are blue, green, yellow, orange, red, indigo, and violet.	S the colours *postmodif [PP* of the rainbow] $V_{copular}$ are C *conj-adj-and* blue green ... violet	

2.12	*Form and Function*		
19. ✓✗	The weather has been very cold just recently	S the weather $V_{copular}$ has been C/A *conj-adjs-implicit-and* very cold just recently	See above
20. ✓✗	Just recently, the weather has been very cold.	{A} *conj-advs-implicit-and* just recently S the weather $V_{copular}$ has been C/A *conj-adjs-implicit-and* very cold	"Just" not interpreted to modify "recently"
	An ADVP may be replace by a different kind of constituent, which is similarly optional and mobile.		
21. ✓✗	The weather has been very cold this month.	S the weather V_{active} has O *implicit-conj-adj* been very cold this *head-noun* month	"been" interpreted as adjective
22. ✓✓	The weather has been very cold during the past week.	S the weather $V_{copular}$ has been C *conj-adjs-implicit-and* very cold *PP* during the past week	"very" not interpreted to modify "cold"
	We cannot always replace a NP by an ADVP or by a PP.		
23. ✓✗	This month has been very cold.	S this month $V_{copular}$ has been C *conj-adjs-implicit-and* very cold	"very" not interpreted to modify "cold"
24. ✓	* Just recently has been very cold.	Failed to parse	
25. ✓	* During the past week has been very cold.	Failed to parse	

2.13	*Clause Structure*		
26. ✓✗	Someone was laughing loudly in the next room.	S someone V was laughing O *PP [P* loudly in] [*NP* the next room]	"loudly" not interpreted as an adverb
27. ✓✗	My mother usually enjoys parties very much.	S my mother V enjoys *adv* usually O parties *postmodif [adj* very] {A} much	"very" is recognized as a postmodifier of "parties"
28. ✓✗	In 1945 the country became totally independent.	{A} *PP* in 1945 S the country $V_{copular}$ became C/A *conj-adjs-implicit-and* totally independent	"totally" not interpreted as modifier of "independent"
29. ✗	I have been in the garden all the time since lunch.	Fragmentary parse	"all the time" not recognized as as an adverbial
	I have been in the garden since lunch.	S [$V_{copular}$ have been C *conj-pps-implicit-and* in the garden since lunch	stative, no adverb
30. ✓✗	Mary gave the visitor a glass of milk.	S Mary V_{active} gave O a glass *postmodif [PP* of milk] O/C/A *iobj* the visitor	"a glass of milk" should be partitive
31. ✓✗	Most people consider these books rather expensive actually.	S most people V_{active} consider O these books C/A *conj-adjs-implicit-and* rather expensive actually	"rather" not interpreted as modifying "expensive" "actually" interpreted as a conjoined adjective, not an adverb
32. ✓✗	You must put all the toys upstairs immediately.	S you V must put O all the toys {A} *conj-advs-implicit-and* upstairs immediately	"upstairs" not recogized as an obligatory adverbial

2.14	*A 'fixed word-order language'*		
33. ✓✗	Usually my mother enjoys parties very much.	S *intensifier* usually *deter* my *head-noun* mother V enjoys O parties *postmodif [adj* very] {A} much	"usually" treated as intensifier of "my mother" and "very" as a postmodifier of "parties"
34. ✓✗	My mother enjoys parties very much, usually.	S my mother V enjoys O parties *postmodif [adj* very] {A} *conjoined-advs-implicit-and* much usually	"very" as a postmodifier of "parties"
35. ✓	* Usually enjoys parties my mother very much.	Failed to parse	
36. ✓	* Enjoys usually my mother parties very much.	Failed to parse	
37. ✓	* My mother parties usually enjoys very much.	Failed to parse	

38. ✓	* Mother my usually enjoys parties much very.	Failed to parse	
2.15	*Adverbials*		
39. ✓✗	My mother enjoys parties very much.	S my mother V enjoys O parties *postmodif* [*adj* very] {A} much	"very" as a postmodifier of "parties"
40. ✗	I have been in the garden all the time since lunch.	Fragmentary parse	See above
41. ✈	* I have been all the time since lunch.	S i $V_{copular}$ have been C/A all the time *postmodif* [*PP* since lunch]	"I have been both parents since he left" would be acceptable
42. ✓✗	You must put all the toys upstairs immediately.	S you V must put O all the toys {A} *conjoined-advs-implicit-and* upstairs immediately	See above
43. ✈	* You must put all the toys immediately.	S you V must put O all the toys {A} immediately	See above
44. ✓✓	To my regret, he refused the offer of help.	{A} *PP* to my regret S he V refused O my offer *postmodif* [*PP* of help]	
45. ✓✗	He was, however, very interested in my other proposals.	S he $V_{copular}$ was C/A *conj-adjs-implicit-and* very interested *PP* in my other proposals {A} *medial-subord-adv* however	"very" not interpreted to modify "interested" ; question of the passive gradient should "interested" be an *adj* or $V_{passive}$?

2.16 – 2.18	*Clause types ; Objects and complements ; Obligatory adverbials*		
46. ✓✓	Someone was laughing.	S someone V was laughing	
47. ✓✓	I have been in the garden.	S i $V_{copular}$ have been C/A *PP* in the garden	
48. ✓✗	You must put all the toys upstairs.	S you V must put O all the toys {A} upstairs	"upstairs" not recognized as an obligatory adverbial
49. ✓✓	My mother enjoys parties.	S my mother V enjoys O parties	
50. ✓✗	Mary gave the visitor a glass of milk.	S Mary V_{active} gave O a glass *postmodif* [*PP* of milk] O/C/A *iobj* the visitor	See above
51. ✓✗	The country became totally independent.	S the country $V_{copular}$ became C/A *conj-adjs-implicit-and* totally independent	"totally" not recognized as modifier of "independent"
52. ✓✗	Most people consider these books rather expensive.	S most people V_{active} consider O these books C/A *conj-adjs-implicit-and* rather expensive	"rather" is not interpreted as a modifier of "expensive"
53. ✓✓	The country became a separate nation.	S the country $V_{copular}$ became C/A a separate nation	
54. ✓✗	Most people considered Picasso a genius.	S most people V_{active} considered O a genius O/C/A *iobj* Picasso	"consider" not recognized as stative, "Picasso" is an iobj
55. ✓✗	He stayed very quiet.	S he $V_{copular}$ stayed C/A *conj-adjs-implicit-and* very quiet	"very" not recognized as a modifier of "quiet"
56. ✓✗	He stayed in bed.	S he V_{active} stayed O *PP* in bed	generates $SV_{copular}C$ (as Quirk suggests) rather that SVO if "stay" is known only as a stative verb
57. ✓✗	They kept him very quiet.	S they V_{active} kept O him C/A *conj-adjs-implicit-and* very quiet	"very" not recognized as a modifier of "quiet"
58. ✓✓	They kept him in bed.	S they V_{active} kept O him O/C/A *PP* in bed	
59. ✓✗	She put the glass down.	S she V_{active} put O the glass *postmodif* [*adj* down]	"down" not recognized as obligatory adverbial
60. ✗	The next meeting will be on the 5th of February.	Fragmentatry parse S the next meeting $V_{copular}$ will be C/A *PP* on *date* February 5	Doesn't recognize "the 5th of February" or "the fifth of February" but can parse "on February 5" as a date
61. ✓✓	The road is under construction.	S the road $V_{copular}$ is C/A *PP* under construction	
62. ✓✓	We kept him off cigarettes.	S we V_{active} kept O him O/C/A *PP* off cigarettes	
63. ✓✗	They treated her kindly.	S they V treated O her {A} kindly	"kindly" not recognized as obligatory adverbial
64. ✓✓	He is without a job.	S he $V_{copular}$ is C/A *PP* without a job	

2.19	*Clause elements subclassified*		
65. ✓✓	Prices rose.	S prices V rose	
66. ✓✓	Elizabeth enjoys classical music.	S Elizabeth V enjoys O classical music	
67. ✓✓	Your face seems familiar.	S your face $V_{copular}$ seems C/A familiar	
68. ✓✓	My sister lives next door.	S my sister V_{active} lives O *ord* next *n* door	Note that Quirk et. al. say this should be $V_{copular}$
69. ✓✗	We all wish you a happy birthday.	S we V wish *adv* all O a happy birthday O/C/A *iobj* you	"all" is an adverb rather than part of the subject
70. ✓✗	The president declared the meeting open.	S the president V declared O the meeting *postmodif* [*adj* open]	"open" is postmodifier of "meeting" rather than a complement

71. ✓✓	The doorman showed the guests into the drawing room.	S the doorman V showed O the guests O/C/A *PP* into the drawing room	

2.21	*Active and passive structures*		
72. ✓✗	They considered him a genius.	S they V_{active} considered O a genius O/C/A *iobj* him	"consider" not copular; "him" is iobj
73. ✓✗	He was considered a genius	$V_{passive}$ was considered O a genius O he	"a genius" should be the passive subject complement
74. ✓✓	He seemed a genius.	S he $V_{copular}$ seemed C/A a genius	
75. ✓✓	A number of people saw the accident.	S a number of people V saw O the accident	Phrasal quantifier misparsed as posmodified noun phrase
76. ✓✗	The accident was seen.	S the accident $V_{copular}$ was C/A seen	should be $V_{passive}$ rather than $V_{copular}$
77. ✓✗	The accident was seen by a number of people.	$S_{passive}$ a number $V_{passive}$ see O *PP* of people O the accident	"of people" is not part of the passive subject
78. ✓✓	John had the book.	S John V_{active} had O the book	
79. ✈	* The book was had by John.	$S_{passive}$ John $V_{passive}$ had O the book	Cannot recognize that some verbs cannot be passivized
80. ✓✓	My father gave me this watch.	S my father V_{active} gave O this watch O/C/A *iobj* me	
81. ✓✓	I was given this watch.	$V_{passive}$ was given O this watch O ¡	
82. ✓✓	I was given this watch by my father.	$S_{passive}$ my father $V_{passive}$ was given O this watch O/C/A ¡	
83. ✓✓	This watch was given to me by my father.	$S_{passive}$ my father $V_{passive}$ gave O *iobj* me O/C/A this watch	
84. ✓✓	This watch was given me.	$V_{passive}$ was given O me O this watch	
85. ✓✗	Queen Victoria considered him a genius.	S queen victoria V_{active} considered O a genius O/C/A *iobj* him	"consider" is known to be stative; should be a copular interpretation
86. ✓✗	He was considered a genius.	$V_{passive}$ was considered O a genius O he	"a genius" should be a complement
87. ✓✗	He was considered a genius by Queen Victoria.	$S_{passive}$ queen victoria $V_{passive}$ considered O a genius O/C/A he	"a genius" should be a complement
88. ✓✗	An intruder must have placed the ladder there.	S an intruder V must have placed O the ladder {A} there	"there" is not interpreted as an object-related adverbial
89. ✓✗	The ladder must have been placed there.	S the ladder $V_{copular}$ must have been C/A *conj-adjs-implicit-and* placed there	$V_{copular}$ rather than $V_{passive}$, "placed" is a complement
90. ✓✗	The ladder must have been placed there by an intruder.	S the ladder $V_{copular}$ must have been C *conj-adjs-implicit-and* placed there *PP* by an intruder	$V_{copular}$ rather than $V_{passive}$; "placed" and "by an intruder" are complements rather than verb and passive subject

2.22	*Copular and complex transitive structures*		
91. ✓✗	I considered her beautiful.	S ¡ V_{active} considered O *det* her *noun-subst* beautiful	should be stative with "beautiful" as complement
92. ✓✗	I considered her to be beautiful.	S ¡ V_{active} considered O [*nom-cl to-inf* be beautiful] O/A/C *iobj* her	should be stative with "to be beautiful" as complement
93. ✓✗	I considered that she was beautiful.	S ¡ V_{active} considered O [*nom-cl full-that-cl* S she $V_{copular}$ was C/A beautiful]	"considered" should be stative

2.23	*Indirect objects and prepositional phrases*		
94. ✓✓	She sent Jim a card.	S she V sent O a card O/C/A *iobj* Jim	
95. ✓✓	She sent a card to Jim.	S she V sent O a card O/C/A *PP* to Jim	
96. ✓✓	She left Jim a card.	S she V left O a card O/C/A *iobj* Jim	
97. ✓✓	She left a card for Jim.	S she V left O a card O/C/A *PP* for Jim	

2.26	*General structural characteristics of phrases*		
98. ✓✗	I went to London.	S ¡ V went O *PP* to London	Prepositional phrase is object rather than adverbial
99. ✈	* I went to.	S ¡ V went *partic* to	"I blew up" would be acceptable
100. ✈	* I went London.	S ¡ V went O London	"I saw London" would be acceptable
101. ✓✗	This is very important indeed	S this V is C *conj-adjs-implicit-and* very important indeed	"very" not modifier of "important"; "indeed" is an adjective
102. ✓✓	This is important.	S this V is C important	

103. ✈	* This is very indeed.	S this V is C *conj-adjs-implicit-and* very indeed	The head is not recognized ; "this is very red" is acceptable

2.27	*Verb phrases and noun phrases*		
104. ✓✓	Jack can play the trombone.	S jack V can play O the trombone	
105. ✓✗	Our team has been beaten.	S our team $V_{copular}$ has been C beaten	$V_{copular}$ rather than $V_{passive}$
106. ✓✓	The room contains vases.	S the room V contains O vases	
107. ✓✓	The room contains some beautiful Flemish vases.	S the room V contains O *deter* some *adj* beautiful *n* vases *premodif* Flemish	
108. ✓✓	The room contains a vase.	S the room V contains O a vase	
109. ✗	* The room contains vase.	S the room V contains O vase	Overgenerates on noun phrases
110. ✓✓	The room contains a beautiful Flemish vase.	S the room V contains O *deter* a *adj* beautiful *n* vase *premodif* Flemish	

2.28	*Summary of phrase structures*		
111. ✓✓	The ship sank.	S the ship V sank	
112. ✓✓	The ship was sinking.	S the ship V was sinking	
113. ✓✗	The ship has been sunk.	S the ship $V_{copular}$ has been C/A sunk	$V_{copular}$ rather than $V_{passive}$
114. ✓✓	The ship must have been sinking.	S the ship V must have been sinking	
115. ✗	The ship may have been being sunk.	Fragmentary parse	Unable to parse "been being"
116. ✓✓	I remember him.	S | V remember O him	
117. ✓✓	I remember Peter.	S | V remember O Peter	
118. ✓✓	I remember Alice's wedding.	S | V remember O *genitive* Alice's *n* wedding	
119. ✓✗	I remember that girl with the red hair.	S | V remember O *deter* that *n* girl O/C/A *PP* with the red hair	"with the red hair" not postmodifier of "girl"
120. ✓✗	I remember all those fine warm days in the country last year.	S | V remember O *predeter* all *deter* those *adj* fine warm *n* days O/C/A in the country last year	"in the country last year" should be postmodifier of "days"
121. ✓✓	I remember a better story than that.	S | V remember O a better story O/C/A *PP* than that	"than that" should complement "better"
122. ✓✓	I remember the best trip that I ever had.	S | V remember O the best trip *postmodif* [*rel-cl-that* S | V had *adv* ever O the best trip]	
123. ✓✓	I remember a good trip that I once had.	S | V remember O a good trip *postmodif* [*rel-cl-that* S | V had *adv* once O a good trip]	
124. ✓✓	The weather was pleasant.	S the weather $V_{copular}$ was C pleasant	
125. ✓✗	The weather was too hot to be enjoyable.	S the weather $V_{copular}$ was C *conj-adjs-implicit-and* too hot *nom-cl to-inf* be enjoyable	"to be enjoyable" should be a postmodifier complement of "too"
126. ✓✗	The weather was incredibly cold.	S the weather $V_{copular}$ was C *conj-adjs-implicit-and* incredibly cold	"incredibly" not interpreted as a modifier of "cold"
127. ✓✗	The weather was pleasant enough.	S the weather $V_{copular}$ was C *conj-adjs-implicit-and* pleasant enough	"enough" not interpreted as a postmodifier of "pleasant"
128. ✓✓	I spoke to him yesterday.	S | V spoke O *PP* to him {A} yesterday	The prepositional phrase is an object
129. ✓✗	I spoke to him quite often.	S | V spoke O *PP* to him {A} *conj-advs-implicit-and* quite often	"quite" not interpreted as modifying "often"
130. ✓✗	I spoke to him severely indeed.	S | V spoke O *PP* to him {A} *conj-advs-implicit-and* severely indeed	"indeed" not interpreted as postmodifier of "severely"
131. ✗	I spoke to him as clearly as I could.	Fragmentary parse	
132. ✓✓	I met her for lunch.	S | V met O her O/C/A *PP* for lunch	
133. ✓✓	I met her at the corner of the street.	S | V met O her O/C/A *PP* at the corner *postmodif* [*PP* of the street]	
134. ✓✓	I met her on Saturday morning.	S | V met O her O/C/A *PP* on Saturday morning	
135. ✓✓	I met her by a strange coincidence.	S | V met O her O/C/A *PP* by strange coincidence	

2.32	*Complementation*		
136. ✓✓	He deceived his father.	S he V deceived O his father	
137. ✈	* He deceived.	S he V deceived	"deceived" is entered into the dictionary as tr_intr

138. ✓✓	He allowed me a respite.	S he V allowed O me O/C/A *iobj* a respite	
139. ✈	* He allowed me.	S he V allowed O me	Doesn't distinguish ditransitive verbs
140. ✓✗	All sales are subject to tax.	S all sales $V_{copular}$ are C/A *n* subject *postmodif* [*PP* to tax]	"subject" is preferred to be a noun. Fails to parse if "subject" is entered only as a verb
141. ✈	* All sales are subject.	S all sales $V_{copular}$ are C/A *n* subject	"All sales are final" would be acceptable
142. ✓✗	Mr_Gould is likely to resign.	S Mr-Gould $V_{copular}$ is C/A likely *nom cl to-inf* resign	"to resign" should complement "likely"
143. ✈	* Mr_Gould is likely.	S Mr-Gould $V_{copular}$ is C/A *adj* likely	
144. ✓✓	Joan was eating her lunch.	S Joan V was eating O her lunch	
145. ✓✓	Joan was eating.	S Joan V was eating	
146. ✓✓	The boat was ready for departure.	S the boat $V_{copular}$ was C/A ready *PP* for departure	This ahould be adjective complementation "for departure"
147. ✓✓	The boat was ready.	S the boat $V_{copular}$ was C/A ready	
148. ✓✓	We always found him cheerful.	S we $V_{copular}$ found *adv* always O him C/A cheerful	
149. ✓✓	He was a cheerful person.	S he $V_{copular}$ was C/A a cheerful person	

2.33	*Modification and complementation*		
150. ✓✗	Greek is a more difficult language than French.	S Greek $V_{copular}$ is C/A *deter* a *quant* more *adj* difficult *n* language *postmodif* [*PP* than French]	"than French" should complement the comparative adverb "more"
151. ✈	* Greek is a difficult language than French.	S Greek $V_{copular}$ is C/A a difficult language *postmodif* [*PP* than French]	"He is a difficult man with a temper" would be acceptable
152. ✓✗	She was too ill to travel.	S she $V_{copular}$ was C/A *conj-adjs-implicit-and* too ill *nom-cl to-inf* travel	"to travel" complements the adverb "too"
153. ✈	* She was ill to travel.	S she $V_{copular}$ was C/A *adj* ill *PP* to travel	"she was ready to travel" would be acceptable
154. ✓✓	She is rather young to drive a car.	S she $V_{copular}$ is C/A *conj-adjs-implicit-and* rather young *nom-cl to-inf* drive a car	
155. ✓✓	She is young to drive a car.	S she $V_{copular}$ is C/A young *nom-cl to-inf* drive a car	

2.43	*Word classes in relation to meaning*		
156. ✓✓	John works hard.	S John V_{active} works {A} hard	
157. ✓✓	John is working hard.	S John V_{active} is working {A} hard	
158. ✓✓	John is tall.	S John $V_{copular}$ is C/A tall	
159. ✗	* John is being tall.		Can't parse "being"
	Adverbs (adjuncts) are dynamic rather than stative in that they add a particular condition of time, place, manner to the dynamic implication of the verb		
160. ✓✓	Marion is beautiful.	S Marion $V_{copular}$ is C/A beautiful	
161. ✓✓	Marion sings beautifully.	S Marion V_{active} sings {A} beautifully	
	Adjectives can take on some of the dynamic implications of the adverb		
162. ✓✓	John is a hard worker.	S John $V_{copular}$ is C/A a hard worker	
163. ✓✓	Marion is a skilful singer.	S Marion $V_{copular}$ is C/A a skilful singer	
164. ✈	* He is knowing English.	S he V_{active} is knowing O English	"he is speaking English" would be acceptable
165. ✗	He is being a nuisance again.	Fragmentary parse	Can't parse "being"
166. ✗	He is being naughty again.	Fragmentary parse	Can't parse "being"

2.44	*Pro-forms*		
167. ✗	John searched the big room and then the small room.	Fragmentary parse	Can't parse "and then the small room"
168. ✗	John searched the big room and then the small one.	Fragmentary parse	Can't parse "and then the small one"
169. ✓✓	The man invited the little Swedish girl because he liked her.	*compound-sent* [the man invited the little Swedish girl] *coord* because [he liked her]	

170. ✓✗	Many students did better than many students expected.	S many students V did O-C *compare-than-good* many students expected	Not clear how the parse tree is SVOC
171. ✓✗	Many students did better than they expected.	S many students V did O-C *compare-than-good* they expected	Not clear how the parse tree is SVOC
172. ✓✓	Mary is in London and John is there too.	*compound-sent* [Mary is in London] *coord* and [John is there too]	
173. ✓✓	Mary arrived on Tuesday and John arrived then too.	*compound-sent* [Mary arrived on Tuesday] *coord* and [John arrived then too]	
174. ✗	The police searched the big room carefully, but the small room less so.	Fragmentary parse	
175. ✓✓	He often behaved prudently, but he did not always behave thus.	*compound-sent* [He often behaved prudently] *coord* but [he did not always behave thus]	
176. ✓✓	He often behaved prudently, but he did not always behave so.	*compound-sent* [He often behaved prudently] *coord* but [he did not always behave so]	
177. ✗	She hoped that they would clean the house carefully before her arrival, but unfortunately they didn't do so.	Timed out after 5 minutes	

2.45	*Wh-words*		
178. ✓✓	Where is Mary?	*be-question wh-word* where O Mary	
179. ✓✓	Mary is in London.	S Mary $V_{copular}$ is C/A *PP* in London	
180. ✗	John is there too.	S John $V_{copular}$ is C/A *conj-adjs-implicit-and* there too	
181. ✗	The place where Mary lives is London.	Fragmentary parse	Can't parse "the place where she lives"
182. ✓✓	I wonder where Mary lives.	S	V wonder O *nom-cl wh-interro-decl-cl* S Mary V lives O where	"where" should be adverbial
183. ✓✓	Where Mary lives, the traffic is very noisy.	{A} *subordinator* where [S Mary V lives] S the traffic $V_{copular}$ is C/A very noisy	
184. ✗	They make him the chairman every year.	Fragmentary parse S they V make O the chairman O/C/A *iobj* him {A} annually	Can't parse "every year" as an adverb ; used "annually"
185. ✗	Who makes him the chairman every year?	Fragmentary parse	Can't parse "every year" or even "annually"
186. ✗	Whom do they make the chairman every year?	Fragmentary parse *wh-question prep-word* whom S they V do make O the chairman {A} annually	Can't parse "every year" as an adverb ; used "annually"
187. ✓✓	What do they make him every year?	*wh-question prep-word* what S they V do make O every year O/C/A *iobj* him	
188. ✓✓	When do they make him the chairman?	*wh-question prep-word* when S they V do make O the chairman O/C/A *iobj* him	
189. ✗	Which cup is yours?	Fragmentary parse	
190. ✓✓	How do you feel?	*wh-question prep-word* how S you V do feel	
191. ✗	How old are you?	Fragmentary parse	

2.47	*Subject and predicate*		
192. ✓✓	Julie buys her vegetables in the market.	S Julie V buys O her vegetables O/C/A *PP* in the market	
193. ✓✓	The train arrived late today.	S the train V arrived {A} *conj-advs-implicit-and* late today	
194. ✓✓	Tigers are carnivorous.	S tigers $V_{copular}$ are C/A carnivorous	
2.48	*Operator and predication*		
195. ✓✓	He had given the girl an apple.	S he V had given O an apple O/C/A *iobj* the girl	
196. ✗	Had he given the girl an apple?	Fragmentary parse	Can't parse "had he given her it?"
197. ✗	John is inviting somebody to dinner.	S John V is inviting O somebody O/C/A *PP* to dinner	
198. ✗	Whom is John inviting to dinner?	Fragmentary parse	
199. ✓✓	The gold has been hidden somewhere.	S the gold $V_{copular}$ has been C/A *conj-adjs-implicit-and* hidden somewhere	
200. ✗	Where has the gold been hidden?	Fragmentary parse	
201. ✓✓	I shall not be working this afternoon.	S	V shall be working *adv* not O this afternoon	"this afternoon" is an object rather than an adverbial
202. ✓✓	We had not given the girl an apple.	S we V had given *adv* not O an apple O/C/A *iobj* the girl	

2.49	*DO, BE, and HAVE as operators*		
203. ✓✓	They often go abroad.	S they V go *adv* often {A} abroad	
204. ✓✗	Do they often go abroad?	*yes-no-question ques-aux-or-modal* do S they V go *adv* often {A} abroad	
205. ✓✗	Her father works in a factory.	S her father V works O *PP* in a factory	Prepositional phrase is an object rather than an adverbial
206. ✓✓	Where does her father work?	*wh-question prep-word* where *ques-aux-or-modal* does S her father V work	Prepositional phrase is an object rather than an adverbial
207. ✓✓	We received your letter.	S we V received O your letter	
208. ✓✓	We did not receive your letter.	S we V did receive *adv* not O your letter	
209. ✓✓	He did it.	S he V did O it	
210. ✓✓	He did not do it	S he V did do *adv* not O it	
211. ✗	Is everything ready?	Fragmentary parse	
212. ✓✓	Everything is ready.	S everything $V_{copular}$ is C/A ready	
213. ✓✓	Was Titian a painter?	*be-question* O Titian O a painter	
214. ✓✓	Titian was a painter.	S Titian $V_{copular}$ was C/A a painter	
215. ✓✗	Are these books for sale?	*be_questioni exist_ref* these O books *postmodif* [*PP* for sale]	"for sale" is postmodifier of books
216. ✓✓	These books are for sale.	S these books $V_{copular}$ are C/A *PP* for sale	
217. ✓✓	Do you have a box of matches?	*yes-no-question ques-aux-or-modal* do S you V have O a box *postmodif* [*PP* of matches]	
218. ✗	Have you a box of matches?	Fragmentary parse	

2.50	*Questions and negation*		
219. ✓✓	Have you borrowed my pencil?	*yes-no-question ques-aux-or-modal* have S you V borrowed O my pencil	
220. ✗	Who has borrowed my pencil?	Fragmentary parse	
221. ✓✓	Why have you borrowed my pencil?	*wh-question prep-word* why *ques-aux-or-modal* have S you V borrowed O my pencil	
222. ✓✓	I haven't borrowed your pencil.	S i V have borrowed *adv* not O your pencil	
223. ✓✓	Did you borrow my pencil?	*yes-no-question ques-aux-or-modal* did S you V borrow O my pencil	
224. ✓✓	Why did you borrow my pencil?	*wh-question prep-word* why *ques-aux-or-modal* did S you V borrow O my pencil	
225. ✓✓	I didn't borrow your pencil.	S i V did borrow *adv* not O your pencil	

2.51	*Predications and pro-forms*		
226. ✓✓	You should eat regularly and take more exercise.	S you V *conj-vps-and* [should eat {A} regularly] [should take O more exercise]	
227. ✓✓	Someone has broken into the house and stolen the money.	S someone V *conj-vps-and* [has broken O *PP* into the house] [has stolen O the money]	Prepositional phrase as object
228. ✓✗	She hoped that he would search the room carefully and he searched the room carefully.	*compound-sent* [S she V hoped O *nom-cl* [*full-that-cl* S he V would search O the room C/A carefully]] *conj* and [S he V searched O the room {A} carefully]	First clause has SVOA but the second has SVO {A}
229. ✓✗	She hoped that he would search the room carefully and he did so.	S she V hoped O *conj-nom-cl-and* [*full-that-cl* S he V would search O the room C/A carefully] [*that-cl* S he V did]] {A} so	"so" not recognized as a pro-form
230. ✓✗	She hoped that he would search the room carefully but he didn't search the room carefully.	*compound-sent* [S she V hoped O *nom-cl* [*full-that-cl* S he V would search O the room C/A carefully]] *conj* but [S he V did search *adv* not O the room {A} carefully]	First clause has SVOA but the second has SVO {A}
231. ✗	She hoped that he would search the room carefully but he didn't do so.	Timed out after 5 minutes	
232. ✓✗	She hoped that he would search the room carefully and he did.	S she V hoped O *conj-nom-cl-and* [*full-that-cl* S he V would search O the room C/A carefully] [*that-cl* S he V did]]	Should be a compound sentence (the stranded operator should be independent)

233. ✓✗	She hoped that he would search the room carefully but he didn't.	S she V hoped O *nom-cl [full-that-cl* S he V would search O the room] {A} *subordinator conj-advs-implicit-and* carefully but [S he V did *adv* not]	Should be a compound sentence (the stranded operator should be independent)
234. ✓	Yes, they do.	{A} yes S they V do	
235. ✓	No, they don't.	{A} no S they V do *adv* not	
236. ✗	Yes, they will.	Fragmentary parse	Can't parse "they will"
237. ✗	No, they won't.	Fragmentary parse	
238. ✗	Yes, they are.	Fragmentary parse	Can't parse "they are"
239. ✓✗	No, they aren't.	{A} no S they $V_{copular}$ are C not	Has "not" as adjective
240. ✓	Yes, they have.	{A} yes S they V have	
241. ✓	No, they haven't.	{A} no S they V have *adv* not	

2.52	*Ellipsis*		
242. ✓✓	Yes, they are paying me for the work.	{A} yes S they V are paying O me O/C/A *PP* for the work	
243. ✓✗	Yes, they are doing so.	{A} yes S they V are doing {A} so	Missed the pro-form
244. ✗	Yes, they are.	Fragmentary parse	Can't parse "they are"

2.54	*Negative forms*		
245. ✓✓	I saw nobody.	S i V saw O nobody	
246. ✓✓	I didn't see anybody.	S i V did see *adv* not O anybody	
247. ✗	Have you never been to London?	Timed out after 5 minutes	
248. ✗	Haven't you ever been to London?	Timed out after 5 minutes	
249. ✓✓	John never invites any students to his parties.	S John V invites *adv* never O any students O/C/A *PP* to his parties	
250. ✓✗	No one ever gives her any encouragement.	S *head-noun* no *postmodif* [*card_num* one] V gives *adv* ever O *det* her *adj* any *head-noun* encouragement	"no" is a head noun unless edited to idiom pronoun "no_one" "her any encouragement" is an object

2.55	*Scope*		
251. ✓✓	Some people never send any Christmas cards.	S some people V send *adv* never O any Christmas cards	
252. ✈	* Any people never send some Christmas cards.	S any people V send *adv* never O some Christmas cards	DIPETT won't know the difference between these two
253. ✓✓	I don't know any people who never send Christmas cards.	S i V do know *adv* not O any people *postmodif* [*rel-cl-who* S any people V send *adv* never O Christmas cards]	
254. ✓✓	Does he seriously believe that?	*yes-no-question ques-aux-or-modal* does S he V believe *adv* seriously O that	
255. ✗	Seriously, does he believe that?	Fragmentary parse	
256. ✓✗	Why doesn't he also take the children abroad?	*wh-question prep-word* why *ques-aux-or-modal* does S *intensifier* not *pers-pron* he V take *adv* also O the children {A} abroad	"not" is part of the subject
257. ✗	Also, why doesn't he take the children abroad?	Timed out after 5 minutes	

2.57	*Directives and exclamations*		
258. ✓✗	Be quiet!	$V_{imperative}$ be C/A quiet	Directives are not distinguished from imperatives
259. ✓✗	Search the room carefully!	$V_{imperative}$ search O the room {A} carefully	
260. ✓✓	Make yourself a cup of tea.	$V_{imperative}$ make O a cup *postmodif* [*PP* of tea] O/C/A *iobj* yourself	
261. ✓✓	Don't hurry!	$V_{imperative}$ do hurry *adv* not	
262. ✓✓	Don't be frightened!	$V_{imperative}$ do be *adv* not C/A frightened	

263. ✓✓	Don't wait for me.	$V_{imperative}$ do wait *adv* not O *PP* for me	
264. ✓	What beautiful clothes she wears!	S what beautiful clothes *postmodif* [*rel-cl-statement* S she V wears O beautiful clothes]	
265. ✓	How well Philip plays the piano!	{A} *conj-advs-implicit-and* how well S Philip V plays O the piano	

2.59	*Grammatical highlighting*		
266. ✓✓	It's Julie who buys her vegetables in the market.	S it $V_{copular}$ is C/A Julie *postmodif* [*rel-cl-who* S Julie V buys O her vegetables O/C/A *PP* in the market	
267. ✓✓	It's her vegetables that Julie buys in the market.	S it $V_{copular}$ is C/A her vegetables *postmodif* [*rel-cl-that* S Julie V buys O her vegetables O/C/A *PP* in the market	
268. ✓✗	It's in the market that Julie buys her vegetables.	S it $V_{copular}$ is C/A *PP* in [the market *postmodif* [*rel-cl-that* S Julie V buys O the market O/C/A her vegetables]]	julie buys the market her vegetables (i.e. her vegetables for the market)
269. ✓✗	Her vegetables Julie buys in the market.	frag her vegetables *postmodif* [*rel-cl-statement* S Julie V buys O her vegetables O/C/A *PP* in the market]	Fronting not recognized
270. ✓✗	In the market Julie buys her vegetables.	{A} in S the market *postmodif* [*appositive* Julie] V buys O her vegetables	Fronting not recognized
271. ✓✓	What you say doesn't matter.	S *nom-cl* [*wh-interro-what* S you V say] V does matter *adv* not	"matter" is head verb
272. ✓✗	It doesn't matter what you say.	S it V does *adv* not O *nom-cl* [*wh-interro-what* S you V say O what] O/C/A *iobj* matter	"matter" is a noun
273. ✗	There was someone knocking at the door.	Fragmentary parse S there $V_{copular}$ was C/A a man *postmodif* [*rel-cl* V knocking O *PP* at the door]	Can't parse "there was someone knocking" ; replaced "someone"

3.1	*Major verb classes*		
274. ✓✓	She left yesterday.	S she V left {A} yesterday	
275. ✓✓	Has she not left yet?	*yes-no-question ques-aux-or-modal* has S she V left *adv* not {A} yet	
276. ✓✓	Did she leave yesterday?	*yes-no-question ques-aux-or-modal* did S she V leave {A} yesterday	
277. ✓✓	She won't leave tomorrow.	S she V will leave *adv* not {A} tomorrow	
278. ✓✗	She might be leaving next week.	S she V might be leaving O next week	"next week" should be an adverb

3.2	*Verb forms and the verb phrase*		
279. ✗	Calling early, she found him at home.	Fragmentary parse	
280. ✗	Having been called early, he felt sleepy all day.	Fragmentary parse	
281. ✓✓	I call regularly.	S i V call {A} regularly	
282. ✓	* He call regularly.	Failed to parse	
283. ✓✓	Call at once.	$V_{imperative}$ call {A} at-once	"at_once" is coded as idiom in DIPETT_0_DICT.PL
284. ✗	They demanded that she call and see them.	Fragmentary parse	DIPETT does not recognize the present subjunctive
285. ✓✓	He may call tonight.	S he V may call {A} tonight	
286. ✓✗	We want her to call.	S we V want O *nom-cl-to-inf* call O/C/A *iobj* her	
287. ✓✗	He calls every day.	S he V calls O every day	"every day" should be an adverb
288. ✓✓	He's calling her now.	S he V is calling O her {A} now	
289. ✗	Calling early, I found her at home.	Fragmentary parse	
290. ✓✓	Someone called yesterday.	S someone V called {A} yesterday	
291. ✓✓	He has called twice today.	S he V has called {A} *conj-advs-implicit-and* twice today	
292. ✓✗	Her brother is called John.	$V_{passive}$ is called O John O her brother	Should be stative, not passive
293. ✗	Called early, he ate a quick breakfast.	Fragmentary parse	

3.21	*Verbs in auxiliary function*		
294. ✓✓	Is he asking any questions?	*be-question* O he O/V asking any questions	
295. ✗	Has he been asking any questions?	Fragmentary parse	
296. ✓✓	Was he asked any questions?	*yes-no-question ques-aux-or-modal* was S he V asked O any questions	
297. ✓✗	Will he be asked any questions?	*yes-no-question ques-aux-or-modal* will S he $V_{copular}$ be C/A *conj-adjs-implicit-and* asked any *head-noun* questions	Should be passive, not stative
298. ✓✓	Has he asked any questions?	*yes-no-question ques-aux-or-modal* has S he V asked O any questions	
299. ✓✓	Does he ask any questions?	*yes-no-question ques-aux-or-modal* does S he V ask O any questions	
	Be and have also have this function as main verbs		
300. ✓✓	Is she a tall girl?	*be-question* O she O/V a tall girl	
301. ✗	Has he any money?	Fragmentary parse	
302. ✗	He might have been being questioned by the police.	Fragmentary parse	Unable to parse "been being"

3.22	*Operator in negation with not*		
303. ✓	* She saw not the play.	S she V saw *adv* not O the play	This interpretation is acceptable
304. ✓	* She not can do it.	Failed to parse	
3.23	*Negative and verb contractions*		
305. ✓✓	She isn't studying.	S she V is studying *adv* not	
306. ✓✓	She's not studying.	S she V has-is studying *adv* not	
307. ✓	* She'sn't studying.	Does not recognize "She'sn't"	
308. ✓	* 'll you be in tonight?	Does not recognize "'ll"	
309. ✓	* No, but I'll tomorrow night.	Failed to parse	
3.24	*Inversion of subject and operator*		
310. ✓✓	Will she come?	*yes-no-question ques-aux-or-modal* will S she V come	
311. ✓	* Plans she to come?	Failed to parse	
312. ✓✓	Does she plan to come?	*yes-no-question ques-aux-or-modal* does S she V plan O *nom-cl-to-inf* come	
313. ✗	At no time was the entrance left unguarded.	Fragmentary parse	

3.26	*Operator in reduced clauses*		
314. ✗	Ann will stay and so will Barbara.	Fragmentary parse	
315. ✓✗	Bill stayed and so did Henry.	S Bill V *conj-verbs-and* [stayed] [did *adv* so] O Henry	This analysis would be fine for "Bill killed and quickly plucked the bird"
316. ✗	Ann won't stay and neither will Barbara.	Fragmentary parse	
317. ✗	Bill didn't stay, nor did Henry.	Fragmentary parse	
318. ✗	Ann will stay late and Barbara will too.	Fragmentary parse	
319. ✓✗	Bill broke his promise, and Henry did too.	S Bill V broke O *nom-cl* [*that-cl-statement* S *conj-nps-and* [his promise] [Henry] V did] {A} too	That-clause construction is incorrect
320. ✗	Ann won't eat much and Barbara won't either.	Fragmentary parse	
321. ✓✗	Bill didn't break his promise, and Henry didn't either.	S Bill V did break *adv* not O *nom-cl* [*that-cl-statement* S *conj-nps-and* [his promise] [Henry] V did *adv* not O either	That-clause construction is incorrect
322. ✗	Ann said she would be late, and late she was.	Fragmentary parse	
323. ✗	Bill said he would win the match, and win the match he did.	Fragmentary parse	
324. ✗	Ann said she would be late, which she was.	Fragmentary parse	
325. ✗	Bill said he would win the match, which he did.	Fragmentary parse	

326. ✗	Ann hoped that we would stay, but unfortunately we couldn't.	Fragmentary parse	
327. ✗	You should take a break whenever you can.	Fragmentary parse	

3.27	*Pre-adverb position*		
328. ✓✓	She never believed his story.	S she V believed *adv* never O his story	
329. ✓✓	She probably believed his story.	S she V believed *adv* probably O his story	
330. ✓✓	She would never believe that story.	S she V would believe *adv* never O that story	
331. ✓✓	She would probably believe that story.	S she V would believe *adv* probably O that story	
332. ✓✓	She would probably never have believed his story.	S she V would have believed *conj-advs-implicit-and* probably never O that story	
333. ✗	She probably never would have believed that story.	Fragmentary parse	
334. ✓✗	She would probably have never believed that story.	S she V would have believed *adv* probably O his story	Parse tree has: advs probably <cr><lf> tokens probably never <cr><lf> tokens never
335. ✈	* She believed never his story.	S she V believed O *intensifier* never *deter* his *head-noun* story	
336. ✈	* She believed probably his story.	S she V believed O *intensifier* probably *deter* his *head-noun* story	
337. ✓✗	She was probably a taxpayer.	S she $V_{copular}$ was C/A *intensifier* probably *deter* a *head-noun* taxpayer	"probably" should be an adverb

3.28	*Quantifier position – quantifiers that modify the subject of the clause may occur after the operator, and before but not after a main verb*		
338. ✓✓	All the boys will be there.	S *predeter* all *deter* the *head-noun* boys $V_{copular}$ will be C/A there	
339. ✓✗	The boys will all be there.	S the boys $V_{copular}$ will be *adv* all C/A there	"all" should be a quantifier
340. ✓✓	Both my parents are working.	S *predeter* both *deter* my *head-noun* parents V are working	
341. ✗	My parents are both working.	Fragmentary parse	
342. ✓✓	All our team played well.	S *predeter* all *deter* our *head-noun* team V played {A} well	
343. ✈	* Our team played all well.	S our team V played O all {A} well	This interpretation is acceptable
344. ✓✓	Each of us owns a bicycle.	S *partitive-deter* each of us V owns O a bicycle	
345. ✓	* We own each a bicycle.	Unable to parse	
346. ✓✗	Our team all played well.	S our team V played *adv* all {A} well	"all" should be a quantifier
347. ✗	We each own a bicycle.	Fragmentary parse	

3.29	*Independence of subject*		
348. ✓✓	The man ought to be here at five.	S the man $V_{copular}$ ought to be C/A *adj* here *PP* at five	
349. ✓✓	The man hopes to be here at five.	S the man V hopes O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* here *PP* at five]	
350. ✓✓	The bus ought to be here at five.	S the bus $V_{copular}$ ought to be C/A *adj* here *PP* at five	
351. ✈	* The bus hopes to be here at five.	S the bus V hopes O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* here *PP* at five]	This sentence is syntactically correct.
352. ✓✗	There used to be a school on the island.	S there V used O *nom-cl* [*to-inf* $V_{copular}$ be C/A a school *postmod* [*PP* on the island]	"used" as a main verb ; does not recognize "used to"
353. ✓	* There hoped to be a school on the island.	S there V hoped O *nom-cl* [*to-inf* $V_{copular}$ be C/A a school *postmod* [*PP* on the island]	This interpretation is acceptable
354. ✓✓	Thousands of people will meet the president.	S thousands *postmod* [*PP* of people] V will meet O the president	
355. ✓✗	The president will be met by thousands of people.	S the president $V_{copular}$ will be C/A *adj* met *PP* by thousands *postmod* [*PP* of people]	Should be passive, not stative
356. ✓✓	Thousands of people hope to meet the president.	S thousands *postmod* [*PP* of people] V hope O *nom-cl* [*to-inf* V meet O the president]	
357. ✓✗	The president hopes to be met by thousands of people.	S the president V hopes O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* met *PP* by thousands *postmod* [*PP* of people]]	"To be met" should be passive, not copular.

3.30	*Additional features of modal auxiliary verbs*		
358. ✓✗	You will be asked questions.	S you $V_{copular}$ will be C/A *adj* asked *head-noun* questions	Should not be stative
359. ✓✓	You ought to comb your hair.	S you V ought to comb O your hair	
360. ✓✓	They might have stolen it.	S they V might have stolen O it	
361. ✓✗	He used to read for hours.	S he V used to read O *PP* for hours	Prepositional phrase should be an adverb, not an object
362. ✓✓	You must write.	S you V must write	
363. ✓✓	She must write.	S she V must write	
364. ✓	* She must writes.	Unable to parse	
365. ✓	* She like to write.	Unable to parse	
366. ✓✓	I think he may retire next May.	S i V think O *nom-cl* [*that-cl* S he V may retire O next May]	
367. ✓✓	I think he might retire next May.	S i V think O *nom-cl* [*that-cl* S he V might retire O next May]	
368. ✓✓	Will you phone him tomorrow?	*yes-no-question ques-aux-or-modal* will S you V phone O him {A} tomorrow	
369. ✓✓	Would you phone him tomorrow?	*yes-no-question ques-aux-or-modal* would S you V phone O him {A} tomorrow	
370. ✗	I told him he must be home early.	Fragmentary parse	
371. ✗	I told him he had to be home early.	Fragmentary parse	

3.31	*The primary verbs BE, HAVE, and DO*		
372. ✓✓	Ann is a happy girl.	S Ann $V_{copular}$ is C/A a happy girl	
373. ✓✓	Is that building a hotel?	*be-question* O that building O/V a hotel	
374. ✓✓	Ann is learning Spanish.	S Ann V is learning O Spanish	
375. ✓✓	The weather has been improving.	S Ann V has been improving	
376. ✓✓	Ann was awarded a prize.	$V_{passive}$ was awarded O a prize O Ann	
377. ✓✗	Our team has never been beaten.	S our team $V_{copular}$ has been *adj* never C/A beaten	Should be passive, not stative

3.33	*HAVE*		
378. ✓✓	I have finished.	S i V have finished	
379. ✓✓	I've not seen her.	S i V have seen *adv* not O her	
380. ✓✓	What has she bought?	*wh-question prep-word* what *ques-aux-or-modal* has S she V bought	
381. ✓✗	They may have been eaten.	S they $V_{copular}$ may have been C/A *adj* eaten	Should be passive, not stative
382. ✓✓	I have no money.	S i V have O *deter* no *head-noun* money	
383. ✓✓	Have you had lunch?	*yes-no-question ques-aux-or-modal* have S you V had O lunch	
384. ✓✗	They have had to sell their car.	S they V have had O *nom-cl* [*to-inf* sell O their car]	Missed semi-auxiliary
385. ✓✓	We don't have any money.	S we V do have *adv* not O any money	
386. ✓✓	Do you have a lighter?	*yes-no-question ques-aux-or-modal* do S you V have O a lighter	
387. ✓✓	We haven't any money.	S we V have *adv* not O any money	
388. ✗	Have you a lighter?	Fragmentary parse	
389. ✓✓	John has got courage.	S John V has got O courage	
390. ✗	* John does have got courage.	S John V does have O *adj* got *noun* courage	Missed "have got"; main verb should not be "have"
391. ✓✓	We haven't got any butter.	S we V have got *adv* not O any butter	
392. ✓✓	We don't have any butter.	S we V do have *adv* not O any butter	
393. ✗	Have you any brothers?	Fragmentary parse	

3.37	*DO*		
394. ✓✗	She doesn't want to stay.	S she V does want *adv* not O *nom-cl* [*to-inf* V stay]	
395. ✓✓	I didn't like mathematics at school.	S i V did like *adv* not O mathematics O/C/A *PP* at school	
396. ✗	Never did he think the book would be finished so soon.	Fragmentary parse	
397. ✓✗	They do want you to come.	S they V do want O *nom-cl* [*to-inf* V come] O *iobj* you	you should not be *iobj*
398. ✗	Michael did say he would be here at nine, didn't he?	Fragmentary parse	
399. ✓✓	Do be quiet.	$V_{copular\text{-}imperative}$ do be C/A quiet	
400. ✗	I haven't done much, I'm afraid.	Fragmentary parse	
401. ✗	I have been meaning to mend that radio but I haven't done it yet.	Timed out after 5 minutes.	
402. ✓✗	Why are you doing that?	*be-question wh-word* why O you O/V doing that	Should be a *wh-* question
403. ✓✗	She didn't earn so much as she might have done.	S she V did earn *adv* not {A} *subordinator* so much as [S she V might have done]	

3.39	*Modal auxiliaries*		
404. ✓✓	He cannot go.	S he V can go *adv* not	
405. ✓✗	He hopes not to go.	S he V hopes *adv* not O *nom-cl* [*to-inf* V go]	
406. ✓✓	He can't go.	S he V can go *adv* not	
407. ✓	* He hopen't go.	Does not recognize "hopen't"	
408. ✓✓	Can we go?	*yes-no-question ques-aux-or-modal* can S we V go	
409. ✓	* Hope we to go?	Unable to parse	
410. ✓✗	Yes, I do hope to come.	{A} yes S i V do hope O *nom-cl* [*to-inf* V come]	
411. ✓	* Yes, I do can come.	Unable to parse	
412. ✗	I can come if you can.	Fragmentary parse	
413. ✗	We can always go early.	Fragmentary parse	
414. ✗	We always hope to go early.	Fragmentary parse	
415. ✓✗	They can all come.	S they V can come *adv* all	"all" should not be an adverb
416. ✓✗	They all hope to come.	S they V hope *adv* all O *nom-cl* [*to-inf* V come]	"all" should not be an adverb
417. ✗	They all can come.	Fragmentary parse	
418. ✓✗	It can be done by Ann.	S it $V_{copular}$ can be C/A *adj* done *PP* by Ann	Should be passive, not stative
419. ✈	* It hopes to be done by him.	S it V hopes O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* done *PP by him*]	"done" should be the head of the to-inf clause
420. ✓✓	I can go.	S i V can go	
421. ✓	* I hope go.	Failed to parse	
422. ✓✓	I want to go.	S i V want O *nom-cl* [*to-inf* V go]	
423. ✓	* I want to can.	Failed to parse	
424. ✓	* I am canning to go.	Failed to parse	
425. ✓✓	I am hoping to go.	S i V am hoping O *nom-cl* [*to-inf* V go]	
426. ✓	* I had canned to go.	Failed to parse	
427. ✓✓	I had hoped to go.	S i V had hoped O *nom-cl* [*to-inf* V go]	
428. ✓✓	She hopes to come.	S she V hopes O *nom-cl* [*to-inf* V come]	
429. ✓	* She cans come.	Failed to parse	

3.42	*dare and need*		
430. ✓✓	He needed to escape.	S he V needed O *nom-cl* [*to-inf* V escape]	
431. ✓✓	He needn't escape.	S he V need escape *adv* not	
432. ✓✓	He doesn't need to escape.	S he V does need *adv* not O *nom-cl* [*to-inf* V escape]	
433. ✓✓	Need we escape?	*yes-no-question ques-aux-or-modal* need S we V escape	
434. ✓✓	Do we need to escape?	*yes-no-question ques-aux-or-modal* do S we V need O *nom-cl* [*to-inf* V escape]	
435. ✓✗	Needn't he escape after_all?	*yes-no-question ques-aux-or-modal* need S *intensifier* not *head-noun* he V escape {A} after-all	"after_all" is modal idiom coded in DIPETT-0-DICT.PL "not" should be an adverb, not an intensifier
436. ✓✓	Dare he not escape?	*yes-no-question ques-aux-or-modal* dare S he V escape *adv* not	
437. ✓✓	Doesn't he need to escape after_all?	*yes-no-question ques-aux-or-modal* does S *intensifier* not *head-noun* he V need O *nom-cl* [*to-inf* V escape] {A} after-all	"not" should be an adverb, not an intensifier
438. ✗	Doesn't he dare to escape?	Fragmentary parse	
439. ✓✓	The king was so hot-tempered that no_one dare tell him what to do.	S the king $V_{copular}$ was C/A *conj-adjs-implicit-and* so hot-tempered *nom-cl* [*full-that-cl* S no-one V dare tell O *nom-cl* [*wh_interro_declarative_clause* $V_{infinitive}$ do O *wh-marker* what] O/C/A *iobj* him]	

3.43	*Ought to*		
440. ✓✓	You ought to stop smoking.	S you V ought to stop O *nom-cl* [*ing-clause* smoking]	
441. ✓✗	You oughtn't to smoke so much.	S you V ought to smoke *adv* not {A} *conj-advs-implicit-and* so much	"so" should be a modifier of much
442. ✗	Ought you to smoke so much?	Fragmentary parse	
443. ✗	Yes, I think I ought to.	Fragmentary parse	
444. ✗	Yes, I think I ought.	Fragmentary parse	

3.43	*Used to*		
445. ✓✓	She used to attend regularly.	S she V used to attend {A} regularly	
446. ✓✗	I used to be interested in birdwatching.	S I V used O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* interested *PP* in birdwatching]	"used" should not be the main verb
447. ✗	He usen't to smoke.	Fragmentary parse	
448. ✓✓	He used not to smoke.	S he V used to smoke *adv* not	
449. ✓✗	He didn't use to smoke.	S he V did use *adv* not O *nom-cl* [*to-inf* V smoke]	"used" should not be the main verb
450. ✓✗	He didn't used to smoke.	S he V did {A} *subordinator* not [*past-part-phrase* $V_{passive}$ used OO *iobj PP* to smoke]	"did" and "use" as main verbs ; "smoke" as noun
451. ✓✗	Did he use to smoke?	*yes-no-question ques-aux-or-modal* did S he V use O *nom-cl* [*to-inf* V smoke]	"use" should not be the main verb
452. ✗	He used to smoke, didn't he?	Fragmentary parse	

3.45	*Modal idioms*		
453. ✓✓	We had better leave soon.	S we V had better leave {A} soon	
454. ✓✗	We'd better leave soon.	S we V had-would leave *adv* better {A} soon	"had better" not recognized
455. ✓✓	I'd rather not say anything.	S I V would rather say *adv* not O anything	
456. ✓✗	They've got to leave immediately.	S they V have got O *nom-cl* [*to-inf* V leave] {A} immediately	"leave" should be the main verb
457. ?	The conference is to take place in Athens.	S the conference V is to take O place O/C/A *PP* in Athens	"take place" ?
458. ✓	* I will have got to leave soon.	S I V will have got O *nom-cl* [*to-inf* V leave] {A} soon	This interpretation is acceptable
459. ?	* The conference has been to take place in Athens.	S the conference $V_{copular}$ has been *nom-cl* [*to-inf* V take O place O/C/A *PP* in Athens]	"take place" ?

460. ✓✗	Hadn't we better lock the door?	*yes-no-question ques-aux-or-modal* had S *intensifier* not *head-noun* we V lock *adv* better O the door	Missed "had better" ; "not" is an intesnifier, not an adverb
461. ✓✗	Would you rather eat in a hotel?	*yes-no-question ques-aux-or-modal* would S you V eat *adv* rather O *PP* in a hotel	Missed "would rather"
462. ✗	We haven't got to pay already, have we?	Timeout at 5 minutes	
463. ✓✓	I wasn't to know that you were waiting.	S ¦ V was to know *adv* not O *nom-cl* [*full-that-cl* S you V were waiting]	
464. ✓✗	I'd rather not stay here all alone.	S ¦ V would rather stay *adv* not O *intensifier* here *pron* all C/A *adj* alone	"here" and "all alone" should be adverbs
465. ✓✗	You'd better not lock the door.	S you V had lock *conj-advs-implicit-and* better not O the door	Missed "had better"
466. ✈	* I'd not rather leave.	S ¦ V would leave *advs* rather not	
467. ✈	* You'd not better resign.	S you V resign *advs* better not	
468. ✓✗	I'd rather rent the cottage.	S ¦ V would rent *adv* rather O the cottage	Missed "would rather"
469. ✈	* The cottage would rather be rented by me.	S the cottage $V_{copular}$ would rather be C/A *adj* rented *postmodif PP* by me	
470. ✓✗	She has got to leave by tomorrow.	S she V has got O *nom-cl* [*to-inf* V leave O *PP-adv* by tomorrow]	Missed "has got to" ; pp_adv as object

3.47	*Semi-auxiliaries – verb idioms with modal meaning and introduced by "have" or "be" – be able to, be about to, be apt to, be bound to, be meant to, be supposed to, be willing to*		
471. ✓✓	Ada isn't going to win.	S Ada V is going to win *adv* not	
472. ✈	* Ada doesn't be going to win.	S Ada $V_{copular}$ does be *adv* not C/A *adj* going *PP* to win	
473. ✓✗	Is Ada going to win?	*be-question* O *head-noun* win *premodif* [*conj-noun-premod-implicit-and* Ada *ing-noun* go]	The whole thing is a noun phrase ; missed "is going to"
474. ✈	* Does Ada be going to win?	*yes-no-question ques-aux-or-modal* does S Ada $V_{copular}$ be C/A *adj* going *PP* to win	
475. ✓	* He is goingn't to. * He is going ton't.	Unable to parse	
476. ✓✓	Brazil is going to win the World Cup.	S Brazil V is going to win O the World Cup	
477. ✓✗	The World Cup is going to be won by Brazil.	S the World Cup V is going O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* won *PP* by Brazil]	Should be passive ; missed "is going to"
478. ✓✗	Several home teams are going to be beaten tomorrow.	S several home teams V are going O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* beaten tomorrow]	Should be passive ; "tomorrow" has been converted to an adjective ; missed "are going to"
479. ✓✗	There are going to be several home teams beaten tomorrow.	S there V are going O *nom-cl* [*to-inf* $V_{copular}$ be C/A several home teams *postmodif* [*adj* beaten]] {A} tomorrow	Missed "are going to"
480. ✈	* Several home teams go to be beaten tomorrow.	S several home teams V go O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* beaten tomorrow]	
481. ✓✗	He was bound to be a failure.	$V_{passive}$ was bound O *nom-cl* [*to-inf* $V_{copular}$ be C/A a failure] O he	Missed "was bound to"
482. ✓✓	Someone bound him to be a failure.	S someone V bound O *nom-cl* [*to-inf* $V_{copular}$ be C/A a failure] O *iobj* him	
483. ✓✗	Someone is going to have to complain.	S someone V is going to have O *nom-cl* [*to-inf* V complain]	Missed "is going to" plus "have to"
484. ✓✗	No_one is likely to be able to recognize her.	S no-one $V_{copular}$ is C/A likely *nom-cl* [*to-inf* $V_{copular}$ be C/A able *nom-cl* [*to-inf* V recognize O her]]	Missed "is likely to" plus "be able to"
485. ✓✗	We haven't been able to solve the problem.	S we $V_{copular}$ have been *adv* not C/A able *nom-cl* [*to-inf* V solve O the problem]	Missed "be able to"
486. ✓	* We haven't could solve the problem.	Unable to parse	
487. ✗	To be allowed to speak freely is a human right.	Fragmentary parse	Can't parse "to be allowed"
488. ✓	* To can speak freely is a human right.	Unable to parse	

3.48	*have to – semi-auxiliary*		
489. ✓✗	I may have to leave early.	Parse tree has "leave" as noun S ¦ V may have O *PP* to leave *postmodif* early	DIPETT's dictionary defines "leave" as both a noun and a verb
490.	* I may have got to leave early.	Parse tree has "leave" as noun S ¦ V may have got O *PP* to leave *postmodif* early	
491. ✓✗	People are having to boil their drinking water during this emergency.	S people V are having O *nom-cl* [*to-inf* V boil O *nom-cl* [*genitive_ing_clause deter* their V drinking O water O/C/A *PP* during this emergency	Missed "are having to" ; "drinking" should be an adjective

492. ✓✗	The administration has had to make unpopular decisions.	S the administration V has had O *nom-cl* [*to-inf* V make O unpopular decisions]	Missed "have to"
493. ✗	These days you must work hard if you want to succeed.	Fragmentary parse	
494. ✓✗	In those days you had to work hard if you wanted to succeed.	*compound-sent* [{A} in S those days *postmodif* [*rel-cl-statement* S you V had *partic* to O those days] V work {A} hard] *coord* if [S you V wanted O *nom-cl* [*to-inf* V succeed]]	Relative clause is incorrect and missed "had to"
495. ✓✓	There must be some solution to the problem.	S there $V_{copular}$ must be C/A some solution *postmodif* [*PP* to the problem]	
496. ✓✗	There had to be some solution to the problem.	S there V had O *nom-cl* [*to-inf* V be C/A some solution *postmodif* [*PP* to the problem]	Missed "had to" ; should be the same analysis as the previous
497. ✗	Do we have to get up early tomorrow.	Fragmentary parse	
498. ✗	Have we to get up early tomorrow?	Fragmentary parse	

3.49	*Catenatives – infinitive following constructions such as: appear to, come to, fail to, get to, happen to, manage to, seem to, tend to, turn out to*		
499. ✓✗	Sam appeared to realize the importance of the problem.	S Sam $V_{copular}$ appeared C/A *nom-cl* [*to-inf* V realize O the importance *postmodif* [*PP* of the problem]]	Missed "appeared to"
500. ✓✗	Sam came to realize the importance of the problem.	S Sam V came O *nom-cl* [*to-inf* V realize O the importance *postmodif* [*PP* of the problem]]	The verb come is in the dictionary as tr_intr missed "came to"
501. ✓✗	Sam failed to realize the importance of the problem.	S Sam V failed O *nom-cl* [*to-inf* V realize O the importance *postmodif* [*PP* of the problem]]	Missed "failed to"
502.	? Sam got to realize the importance of the problem.	S Sam V got O *nom-cl* [*to-inf* V realize O the importance *postmodif* [*PP* of the problem]]	
503. ✓✗	Sam seemed to realize the importance of the problem.	S Sam $V_{copular}$ seemed C/A *nom-cl* [*to-inf* V realize O the importance *postmodif* [*PP* of the problem]]	Missed "seemed to"
504. ✓✗	Sam didn't appear to realize the importance of the problem.	S Sam $V_{copular}$ did appear *adv* not C/A *nom-cl* [*to-inf* V realize O the importance *postmodif* [*PP* of the problem]]	Missed "appeared to"
505. ✓✗	Sam didn't come to realize the importance of the problem.	S Sam V did come *adv* not O *nom-cl* [*to-inf* V realize O the importance *postmodif* [*PP* of the problem]]	Missed "come to"
506. ✓✗	The importance of the problem appeared to be realized by Sam.	S the importance *postmodif* [*PP* of the problem]] $V_{copular}$ appeared C/A *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* realized *PP* by Sam]]	Should be passive ; missed "appeared to"
507. ✓✗	The importance of the problem came to be realized by Sam.	S the importance *postmodif* [*PP* of the problem]] V came O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* realized *PP* by Sam]]	Should be passive ; missed "came to"
508. ✈	? The importance of the problem failed to be realized by Sam.	S the importance *postmodif* [*PP* of the problem]] V failed O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* realized *PP* by Sam]]	Should be passive ; missed "failed to"
509. ✈	* The importance of the problem got to be realized by Sam.	S the importance *postmodif* [*PP* of the problem]] V got O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* realized *PP* by Sam]]	Should be passive ; missed "got to"
510. ✓✗	The importance of the problem seemed to be realized by Sam.	S the importance *postmodif* [*PP* of the problem]] $V_{copular}$ seemed C/A *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* realized *PP* by Sam]]	Should be passive ; missed "seemed to"
511. ✓✗	John attempted to attack the burglar.	S John V attempted O *nom-cl* [*to-inf* V attack O the burglar]	Missed "attempted to"
512. ✓✗	John attempted an attack on the burglar.	S John V attempted O the burglar O/C/A *iobj* an attack *postmodif* [*adj* on]	"on" as an adjective ; this is a very strange interpretation
513. ✓✗	John appeared to attack the burglar.	S John V appeared O *nom-cl* [*to-inf* V attack O the burglar]	Missed "appeared to"
514. ✈	* John appeared an attack on the burglar.	S John $V_{copular}$ appeared C/A an attack *postmodif PP* on the burglar	This interpretation is acceptable
515. ✓✗	The girl started out working.	S the girl V started O *nom-cl* [*ing-cl* V working *adv* out]	Missed "started out" progressive construction
516. ✓✗	The girl kept working.	S the girl V kept O *nom-cl* [*ing-cl* V working]	Missed "kept" progressive construction
517. ✓✗	The girl kept on working.	S the girl V kept O *nom-cl* [*ing-cl* V working *adv* on]	Missed "kept on" progressive construction
518. ✓✗	The girl went on working.	S the girl V went O *nom-cl* [*ing-cl* V working *adv* on]	Missed "went on" progressive construction
519. ✗	Our team got beaten by the visitors.	Parsed as a single noun fragment	Failed to recognize "get" passive auxiliary

3.53	*Nonfinite verb phrases*		
520. ✓✓	He smokes.	S he V smokes	
521. ✓✗	To smoke like that must be dangerous.	*nom-cl [to-inf* V smoke O *PP* like that] $V_{copular}$ must be C/A dangerous	"like that" should be an adverb
522. ✓✓	Mary is having a smoke.	S Mary V is having O a smoke	
523. ✓✗	I regret having started to smoke.	S i V regret O *ing-noun* [V having *postmodif [rel-cl-predicate* $V_{passive}$ started O *iobj PP* to *n* smoke O *ing-noun* V having]]	This should be a nominal clause
524. ✓✗	He must smoke forty a day.	S he V must smoke O a day O *iobj* forty	"a day" should be an adverb
525. ✗	The cigars smoked here tend to be expensive.	Fragmentary parse	
526. ✓✗	You have been smoking all day.	S you V have been smoking O all day	"all day" should be an adverb
527. ✗	That was the last cigarette to have been smoked by me.	Fragmentary parse	

3.54	*Simple and complex verb phrases*		
528. ✗	Work harder!	Fragmentary parse	Is only able to parse "work hard."
529. ✗	It is important that he work hard.	Fragmentary parse	Subjunctive not recognized.
530. ✓✓	He may have examined it.	S he V may have examined O it	
531. ✓✓	He may be examining it.	S he V may be examining O it	
532. ✓✗	He may be examined.	S he $V_{copular}$ may be C/A *adj* examined	Should be passive, not stative
533. ✓✓	He has been examining it.	S he V has been examining O it	
534. ✓✗	He has been examined.	S he $V_{copular}$ has been C/A *adj* examined	Should be passive, not stative
535. ✗	He is being examined.	Fragmentary parse	Unable to parse "be being"
536. ✓✓	He may have been examining it.	S he V may have been examining O it	
537. ✓✗	He may have been examined.	S he $V_{copular}$ may have been C/A *adj* examined	Should be passive, not stative
538. ✗	He may be being examined.	Fragmentary parse	Unable to parse "be being"
539. ✗	He has been being examined.	Fragmentary parse	Unable to parse "be being"
540. ✗	He may have been being examined.	Fragmentary parse	Unable to parse "be being"

3.56	*Simple and complex nonfinite verb phrases*		
541. ✓✓	Smoking cigarettes is dangerous.	S *nom-cl [ing-cl* V smoking O cigarettes] $V_{copular}$ is C/A dangerous	
542. ✓✓	They caught him smoking cigarettes.	S they V caught O *nom-cl [ing-cl* V smoking O cigarettes] O *iobj* him	
543. ✗	We're glad to have invited you.	Fragmentary parse	
544. ✓✗	I'd like to be working.	S i V would like O *nom-cl [to-inf* $V_{copular}$ be C/A *adj* working]	"to be working" should be the progresive of "work", not stative
545. ✓✗	I'd hate to be questioned about it.	S i V would hate O *nom-cl [to-inf* $V_{copular}$ be C/A *adj* questioned *PP* about it]	"to be questioned" should not be stative
546. ✓✗	I'm glad to have been working.	S i $V_{copular}$ am C/A *adj* glad *nom-cl [to-inf* $V_{copular}$ have O *adj* been *ing-noun* [V working]]	"been working" should not be object of "have"
547. ✗	He's said to have been invited.	Fragmentary parse	
548. ✗	I expected to be being interviewed then.	Fragmentary parse	Can't parse "be being"
549. ✗	Having invited you, I expected you to come.	Fragmentary parse	
550. ✗	You can probably get an extension on the grounds of being teaching.	Fragmentary parse	
551. ✓✓	When questioned, he denied everything.	*initial subordinate nom-cl [wh-interro-decl-cl* S *wh-marker* when V questioned] S he V denied O everything	
552. ✓✗	Having been working all day, I'm very tired.	*initial subordinate nom-cl [ing-cl* V having O *conj-adjp-implicit-and* been working all *head-noun* day] S i $V_{copular}$ am C/A *conj-adjp-implicit-and* very tired	"having" should not be a main verb

553. ✗	Having been invited, he should have come.	Fragmentary parse	
554. ✗	I saw her being interviewed.	Fragmentary parse	

3.57	*Gradience between one and two verb phrases*		
555. ✗	We had hoped to have finished by then.	Fragmentary parse	
556. ✓✗	I am hoping to be seeing her tomorrow.	S I V am hoping O *nom-cl* [*to-inf* $V_{copular}$ be C/A *nom-cl* [*ing-cl* V seeing O her] {A} tomorrow	"to be seeing" should be progressive, not copular
557. ✗	? They must have been expecting to have been being paid.	Fragmentary parse	Can't parse "be being"
558. ✓✓	Sarah and I are going to be leaving tonight.	S *conj-nps-and* Sarah i V are going to be leaving {A} tonight	
559. ✓✗	The walls were supposed to be repainted.	$V_{passive}$ were supposed O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* repainted] O the walls	Missed the semi-auxiliary "be supposed to"
560. ✗	If you'd done that, we would have had to have arrested you.	Fragmentary parse	

3.59	*The mandative subjunctive*		
561. ✗	The committee proposes Mr_Day be elected.	Fragmentary parse	Can't parse "Mr." and subjunctive not recognized
562. ✗	The committee proposes that Mr_Day be elected.	Fragmentary parse	Can't parse "Mr." and subjunctive not recognized
563. ✗	The committee proposed that Mr_Day be elected.	Fragmentary parse	Can't parse "Mr." and subjunctive not recognized
564. ✗	I demand that the committee reconsider its decision.	Fragmentary parse	Subjunctive not recognized
565. ✗	His sole requirement was that the system work.	Fragmentary parse	Subjunctive not recognized
566. ✗	They recommend that this tax be abolished.	Fragmentary parse	Subjunctive not recognized
567. ✗	It is appropriate that this tax be abolished.	Fragmentary parse	Subjunctive not recognized
568. ✗	We were faced with the demand that this tax be abolished.	Fragmentary parse	Subjunctive not recognized
569. ✓✗	They insisted that we not eat meat.	S They V insisted O *nom-cl* [*full-that-cl* S we V eat *adv* not O meat]	Should be subjunctive, not indicative
570. ✓✓	They insisted that we do not eat meat.	S They V insisted O *nom-cl* [*full-that-cl* S we V do eat *adv* not O meat]	Indicative is correct

3.61	*Uses of the present subjunctive*		
571. ✗	Even if that be the official view, it cannot be accepted.	Fragmentary parse	Subjunctive not recognized
572. ✗	If that be the official view, it cannot be accepted.	Fragmentary parse	Subjunctive not recognized
573. ✗	The President must reject this proposal, lest it cause strife and violence.	Fragmentary parse	Subjunctive not recognized

3.62	*The were-subjunctive – introduced by conjunctions such as: if, as if, as though, though and also in nominal clauses after verbs such as: wish, suppose*		
574. ✗	If I were rich, I would buy you anything you wanted.	Fragmentary parse	
575. ✗	If I was rich, I would buy you anything you wanted.	Fragmentary parse	
576. ✓✗	Tim always speaks quietly on the phone, as_though he were telling a secret.	S Tim V speaks *adv* always O *PP* quietly on the phone {A} *subordinator* as-though [S he $V_{copular}$ were C/A *nom-cl* [*ing-cl* V telling O a secret]	"were" should be subjunctive rather than copular ; "quietly on the phone" should be an adverb
577. ✓✗	Tim always speaks quietly on the phone, as_though he was telling a secret.	S Tim V speaks *adv* always O *PP* quietly on the phone {A} *subordinator* as-though [S he V was telling O a secret]	"quietly on the phone" should be an adverb
578. ✓✗	I wish the journey were over.	S I V wish O *nom-cl* [*that-cl* S the journey $V_{copular}$ were C/A over]	Parsed via a "that" construction rather than the subjunctive

3.64	*Voice*		
579. ✓✓	She misses it.	S she V_{active} misses O it	present_simple
580. ✓✓	She missed it.	S she V_{active} missed O it	past_simple
581. ✓✓	She may miss it.	S she V_{active} may miss O it	may_present
582. ✓✓	She has missed it.	S she V_{active} has missed O it	present_perfect_simple
583. ✓✓	She is missing it.	S she V_{active} is missing O it	present_continuous
584. ✓✓	She may have missed it.	S she V_{active} may have missed O it	may_perfect
585. ✓✓	She may be missing it.	S she V_{active} may be missing O it	may_present_continuous
586. ✓✓	She has been missing it.	S she V_{active} has been missing O it	present_perfect_continuous
587. ✓✓	She may have been missing it.	S she V_{active} may have been missing O it	may_perfect_continuous
588. ✓✗	She is missed.	S she $V_{copular}$ is C/A *adj* missed	Should be passive, not stative
589. ✓✗	She was missed.	S she $V_{copular}$ was C/A *adj* missed	Should be passive, not stative
590. ✓✗	She may be missed.	S she $V_{copular}$ may be C/A *adj* missed	Should be passive, not stative
591. ✓✗	She has been missed.	S she $V_{copular}$ has been C/A *adj* missed	Should be passive, not stative
592. ✗	She is being missed.	Fragmentary parse	Can't parse "be being"
593. ✓✗	She may have been missed.	S she $V_{copular}$ may have been C/A *adj* missed	Should be passive, not stative
594. ✗	She may be being missed.	Fragmentary parse	Can't parse "be being"
595. ✗	She has been being missed.	Fragmentary parse	Can't parse "be being"
596. ✗	She may have been being missed.	Fragmentary parse	Can't parse "be being"

3.66	*The passive auxiliaries: be and get*		
597. ✗	The cat got run over.	Fragmentary parse	"get" is not recognized as a passive auxiliary
598. ✓✗	James got beaten last night.	S James V got O *ord* beaten last *head-noun* night	"beaten last" is parsed as an ordinal; "last night" not parsed as adverb
599. ✗	James got caught by the police.	Fragmentary parse	
600. ✓✗	We are getting bogged down in all sorts of problems.	S we V are getting O *PP* bogged down in *predeter* all *head-noun* sorts *postmodif* [*PP* of problems]	"get" is not recognized as a passive auxiliary
601. ✗	I have to get dressed before eight o'clock.	Fragmentary parse	
602. ✗	I don't want to get mixed up with the police again.	Fragmentary parse	
603. ✗	You argument gets a bit confused here.	Fragmentary parse	

3.68	*Verb constraints: some verbs can be passivized, others cannot. Some verbs and verb constructions appear only in the passive*		
604. ✓✓	The coat does not fit you.	S the coat V does fit *adv* not O you	
605. ✈	* You are not fitted by the coat.	S you $V_{copular}$ are *adv* not C/A *adj* fitted *PP* by the coat	DIPETT does not recognize the passive, let alone that "fit" cannot be passivized
606. ✗	The police want him.	Fragmentary parse	"the police" is not recognized as being plural
607. ✓✓	He is wanted by the police.	$S_{passive}$ the police $V_{passive}$ want O he	
608. ✓✓	Betty was said to be a good teacher.	$V_{passive}$ was said O *nom-cl* [*to-inf* $V_{copular}$ be C/A a good teacher] O Betty	
609. ✈	* They said her to be a good teacher.	S they V said O *nom-cl* [*to-inf* $V_{copular}$ be C/A a good teacher] O her	This is acceptable for DIPETT
610. ✓✗	He was born in Tubingen.	S He $V_{copular}$ was C/A *adj* born *PP* in Tubingen	Should be passive, not copular
611. ✈	? His mother bore him in Tubingen.	S His mother V bore O him O/C/A *PP* in Tubingen	This is acceptable for DIPETT
612. ✓✗	The wanted man fell into the water and was drowned.	S the wanted man V *conj-vps-and* [V fell O *PP* into the water] [$V_{copular}$ was C/A *adj* drowned]	"was drowned" should be passive, not copular

3.69	*Prepositional verbs*		
613. ✓✗	The engineers went very carefully into the problem.	S the engineers V went O *PP* very carfully into the problem	"went into" not recognized as prepositional verb
614. ✓✗	They eventually arrived at the expected result.	S they V arrived *adv* eventually O *PP* at the expected result	"arrived at" not recognized as prepositional verb
615. ✓✗	The engineers went very carefully into the tunnel.	S the engineers V went O *PP* very carfully into the tunnel	"carefully into the tunnel" should be an adverb
616. ✓✗	They eventually arrived at the splendid stadium.	S they V arrived *adv* eventually O *PP* at the splendid stadium	"at the splendid stadium" should be an adverb
617. ✓✗	The problem was very carefully gone into by the engineers.	S the problem $V_{copular}$ was C/A *adj* very {A} *subordinator* carefully [*past-part-phrase* $S_{passive}$ the engineers $V_{passive}$ gone *partic* into O *main-cl-subj*]	"by" got lost ; "gone into" should be the main verb
618. ✗	The expected result was eventually arrived at.	Fragmentary parse	

3.70	*Object constraints*		
619. ✓✓	John thought that she was attractive.	S John V thought O *nom-cl* [*full-that-cl* S she $V_{copular}$ was C/A attractive]	
620. ✓✓	? That she was attractive was thought by John.	S John $V_{passive}$ thought O *nom-cl* [*full-that-cl* S she $V_{copular}$ was C/A attractive]	
621. ✓✓	John hoped to meet her.	S John V hoped O *nom-cl* [*to-inf* V meet O her]	
622. ✓✓	? To meet her was hoped by John.	S John $V_{passive}$ hoped O *nom-cl* [*to-inf* V meet O her]	
623. ✓✓	John enjoyed seeing her.	S John V enjoyed O *nom-cl* [*ing-cl* V seeing O her]	
624. ✓✓	? Seeing her was enjoyed by John.	$S_{passive}$ John $V_{passive}$ enjoyed O *nom-cl* [*ing-cl* V seeing O her]	

3.74	*The passive gradient*		
	Central passives have a direct active-passive relation.		
625. ✓✓	This violin was made by my father.	$S_{passive}$ my father $V_{passive}$ made O this violin	
626. ✓✗	This conclusion is hardly justified by the results.	S this conclusion $V_{copular}$ is C/A *conj-adjs-implicit-and* hardly justified *PP* by the results	Should be passive, not copular
627. ✓✗	Coal has been replaced by oil.	S coal $V_{copular}$ has been C/A *adj* replaced *PP* by oil	Should be passive, not copular
628. ✓✗	This difficulty can be avoided in several ways.	S this difficulty $V_{copular}$ can be C/A *adj* avoided *PP* in several ways	Should be passive, not copular
	Semi-passives represent a mixed or semi-passive class whose members have both verbal and adjectival properties.		
629. ✓?	We are encouraged to go on with the project.	S we $V_{copular}$ are C/A *adj* encouraged *nom-cl* [*to-inf* V go O *PP* on with the project]	This interpretation is acceptable
630. ✓?	Leonard was interested in linguistics.	$V_{passive}$ was interested O *iobj* in linguistics O Leonard	This alternative is also acceptable
	Psuedo-passives have neither an active transform nor a possibility of agent addition		
631. ✓✓	The building is already demolished.	S the building $V_{copular}$ is C/A *conj-adjs-implicit-and* already demolished	
632. ✓✗	The modern world is getting more highly industrialized and mechanized.	S the modern world V is getting *nom-cl* [*that-cl* S more V *conj-verbs-and* [industrialized *adv* highly] [mechanized *adv* highly]	"get" not recognized as a passive auxilliary
	The following can be clearly analyzed as having an adjectival complement following a copular verb, the possibility of inserting very confirms the adjectival status of tired.		
633. ✓✓	My uncle was very tired.	S my uncle $V_{copular}$ was C/A *conj-adjs-implicit-and* very tired	
634. ✗	My uncle got tired.	Fragmentary parse	"tired" is entered only as a past participle
635. ✓✓	My uncle seemed tired.	S my uncle $V_{copular}$ seemed C/A tired	

5.1	*Types of noun phrase*		
636. ✓✓	The girl is Mary Smith.	S the girl $V_{copular}$ is C/A *propernoun* Mary *appositive* Smith	Smith should not be an appositive of Mary
637. ✓✓	The blonde girl is Mary Smith.	S the girl *premodif* blonde $V_{copular}$ is C/A *propernoun* Mary *appositive* Smith	
638. ✓✓	The blonde girl in blue jeans is Mary Smith.	S the girl *premodif* blonde *postmodif PP* in blue jeans $V_{copular}$ is C/A *propernoun* Mary *appositive* Smith	
639. ✓✓	The blonde girl wearing blue jeans is Mary Smith.	S the girl *premodif* blonde *postmodif* [*rel-cl* V wearing O blue jeans] $V_{copular}$ is C/A *propernoun* Mary *appositive* Smith	

640. ✓✓	The blonde girl who is wearing blue jeans is Mary Smith.	S the girl *premodif* blonde *postmodif* [*rel-cl-who* V wearing O blue jeans] $V_{copular}$ is C/A *propernoun* Mary *appositive* Smith	
641. ✓✓	She is Mary Smith.	S she $V_{copular}$ is C/A *propernoun* Mary *appositive* Smith	
642. ✓✓	* The blonde she is Mary Smith.	Fragmentary parse	
643. ✓✓	? She in blue jeans is Mary Smith	Fragmentary parse	

5.2	*Noun classes: count, non-count, and proper nouns*		
	Proper nouns		
644. ✓✓	I saw Matthew.	S ¦ V saw O *proper_noun sg3* Matthew	
645. ✓	* I saw the Matthew.	Fragmentary parse	
646. ✓	* I saw a Matthew.	Fragmentary parse	
647. ✈	* I saw some Matthew.	S ¦ V saw O *adj* some *proper_noun sg3* Matthew	"some" is entered into DIPETT's dictionary as a pronoun, a determiner, and as an adverb
648. ✓	* I saw Matthews.	"Matthews" not recognized.	
	Count nouns: individual countable entities		
649. ✈	* I saw book.	S ¦ V saw O *deter* nil *countnoun sg3* book	This sentence is ungrammatical and should fail
650. ✓✓	I saw the book.	S ¦ V saw O *deter* the *countnoun sg3* book	
651. ✓✓	I saw a book.	S ¦ V saw O *deter* a *countnoun sg3* book	
652. ✓?	* I saw some book.	S ¦ V saw O *deter* some *countnoun sg3* book	As a determiner, "some" may occur with singular count nouns, meaning "certain"
653. ✓✓	I saw books.	S ¦ V saw O *deter* nil *countnoun pl* book	
	Mass nouns: non-count nouns		
654. ✓✓	I saw furniture.	S ¦ V saw O *deter* nil *massnoun sg3* furniture	
655. ✓✓	I saw the furniture.	S ¦ V saw O *deter* the *massnoun sg3* furniture	
656. ✓	* I saw a furniture.	Fragmentary parse	
657. ✓✓	I saw some furniture.	S ¦ V saw O *deter* some *massnoun sg3* furniture	
658. ✓	* I saw furnitures.	"furnitures" not recognized.	
	Nouns that are both count and non-count		
659. ✓✓	I saw brick.	S ¦ V saw O *deter* nil *countnoun sg3* brick	
660. ✓✓	I saw the brick.	S ¦ V saw O *deter* the *countnoun sg3* brick	
661. ✓✓	I saw a brick.	S ¦ V saw O *deter* a *countnoun sg3* brick	
662. ✓✓	I saw some brick.	S ¦ V saw O *deter* some *countnoun sg3* brick	
663. ✓✓	I saw bricks.	S ¦ V saw O *deter* nil *countnoun pl* brick	

Appendix C: DIPETT v3.0 Performance on the TSNLP Phenomena

Appendix C shows DIPETT v3.0's performance on the detailed TSNLP phenomena.

Phenomenon	*Number of Sentences*	*Bad Data*	*Correct Full Parses*	*Incorrect Full Parses*	*Fragmentary Parses*
C_Agreement-Collectives	3	1	0	2	0
C_Agreement-Coordinated_subj(correlatives)	14	2	6	6	0
C_Agreement-NP_V	2	0	0	2	0
C_Agreement-Parentheticals	6	0	3	3	0
C_Agreement-Partitives	20	3	11	5	1
C_Agreement-Pron_V	7	0	0	7	0
C_Agreement-Pron_V(be)	7	0	0	7	0
C_Agreement-Relative_clauses	8	0	1	7	0
C_Complementation-Divalent-Complementizer(0)_indicative	1	0	0	1	0
C_Complementation-Divalent-Complementizer(0)_indicative-Subj(it)	1	0	0	1	0
C_Complementation-Divalent-Complementizer(that)_indicative	1	0	0	1	0
C_Complementation-Divalent-Complementizer(that)_indicative-Subj(it)	1	0	0	1	0
C_Complementation-Divalent-Complementizer(that)_subjunctive/should_bare_infinitive	2	0	1	1	0
C_Complementation-Divalent-Complementizer(WH)	1	0	0	1	0
C_Complementation-Divalent-Direct_object	1	0	0	1	0
C_Complementation-Divalent-Particle	1	0	0	0	1
C_Complementation-Divalent-PP	28	0	0	28	0
C_Complementation-Divalent-Predicate	3	0	0	3	0
C_Complementation-Divalent-VP_bare_infinitive	1	0	0	1	0
C_Complementation-Divalent-VP_ing	1	0	0	1	0
C_Complementation-Divalent-VP_to_infinitive	1	0	0	1	0
C_Complementation-Equi-Obj_controlled	4	0	0	0	4
C_Complementation-Equi-Pobj_controlled	2	0	2	0	0
C_Complementation-Equi-Subj_controlled	3	0	0	0	3
C_Complementation-Monovalent-Subj(it)	1	0	0	1	0
C_Complementation-Monovalent-Subj(NP)	1	0	0	1	0
C_Complementation-Raising-Subj_to_obj-VP_bare_infinitive	3	0	3	0	0
C_Complementation-Raising-Subj_to_obj-VP_to_infinitive	8	0	1	0	7
C_Complementation-Raising-Subj_to_subj	6	0	0	0	6
C_Complementation-Tetravalent	2	0	1	0	1
C_Complementation-Trivalent-Direct_object_as_AP	1	0	0	0	1
C_Complementation-Trivalent-Direct_object_as_NP	1	0	0	0	1
C_Complementation-Trivalent-Direct_object_as_VP_ing	1	0	1	0	0
C_Complementation-Trivalent-Direct_object_particle	32	0	0	32	0
C_Complementation-Trivalent-Direct_object_PP	1	0	0	1	0

Phenomenon	Number of Sentences	Bad Data	Correct Full Parses	Incorrect Full Parses	Fragmentary Parses
C_Complementation-Trivalent-Direct_object_predicate(AP)	1	0	0	1	0
C_Complementation-Trivalent-Direct_object_predicate(NP)	1	0	0	0	1
C_Complementation-Trivalent-Direct_object_predicate(PP)	1	0	0	1	0
C_Complementation-Trivalent-Direct_object_VP_bare_infinitive	12	0	0	0	12
C_Complementation-Trivalent-Direct_object_VP_ed	1	0	0	0	1
C_Complementation-Trivalent-Direct_object_VP_ing	1	0	0	0	1
C_Complementation-Trivalent-Indirect_object_complementizer(that)_indicative	1	0	0	0	1
C_Complementation-Trivalent-Indirect_object_complementizer(WH)_indicative	8	0	0	2	6
C_Complementation-Trivalent-Indirect_object_complementizer(WH)_VP_to_infinitive	1	0	0	1	0
C_Complementation-Trivalent-Indirect_object_direct_object	1	0	0	1	0
C_Complementation-Trivalent-Indirect_object_PP	1	0	0	1	0
C_Complementation-Trivalent-Indirect_object_prep_VP_ing	2	0	0	2	0
C_Complementation-Trivalent-Indirect_object_VP_to_infinitive	1	0	0	1	0
C_Complementation-Trivalent-Particle_PP	1	0	0	0	1
C_Complementation-Trivalent-PP_complementizer(0)_indicative	1	0	1	0	0
C_Complementation-Trivalent-PP_complementizer(that)_indicative	1	0	0	0	1
C_Complementation-Trivalent-PP_complementizer(WH)_indicative	1	0	0	0	1
C_Complementation-Trivalent-PP_complementizer(WH)_VP_to_infinitive	1	0	0	1	0
C_Complementation-Trivalent-PP_PP	1	0	0	0	1
C_Complementation-Trivalent-PP_VP_to_infinitive	1	0	0	0	1
C_Diathesis-Active	1	0	0	1	0
C_Diathesis-Active-Divalent-Complementizer(0)_indicative	1	0	1	0	0
C_Diathesis-Active-Divalent-Complementizer(0)_indicative-Subj(it)	2	0	2	0	0
C_Diathesis-Active-Divalent-Complementizer(that)_indicative	1	0	1	0	0
C_Diathesis-Active-Divalent-Complementizer(that)_subjunctive/should_bare_infinitive	2	0	2	0	0
C_Diathesis-Active-Divalent-Direct_object	4	0	4	0	0
C_Diathesis-Active-Divalent-Particle	1	0	1	0	0
C_Diathesis-Active-Divalent-PP	3	0	3	0	0
C_Diathesis-Active-Divalent-Predicate	1	0	1	0	0
C_Diathesis-Active-Divalent-VP_bare_infinitive	1	0	1	0	0
C_Diathesis-Active-Divalent-VP_ing	2	0	2	0	0
C_Diathesis-Active-Divalent-VP_to_infinitive	1	0	1	0	0
C_Diathesis-Active-Equi-Obj_controlled	2	0	2	0	0
C_Diathesis-Active-Monovalent-Subj(it)	1	0	1	0	0
C_Diathesis-Active-Monovalent-Subj(NP)	1	0	1	0	0
C_Diathesis-Active-Raising-Subj_to_obj-VP_to_infinitive	2	0	2	0	0
C_Diathesis-Active-Tetravalent	2	0	2	0	0
C_Diathesis-Active-Trivalent-Direct_object_as_AP	1	0	1	0	0
C_Diathesis-Active-Trivalent-Direct_object_as_NP	1	0	1	0	0
C_Diathesis-Active-Trivalent-Direct_object_as_VP_ing	1	0	1	0	0
C_Diathesis-Active-Trivalent-Direct_object_particle	2	0	2	0	0
C_Diathesis-Active-Trivalent-Direct_object_PP	2	0	2	0	0
C_Diathesis-Active-Trivalent-Direct_object_predicate(AP)	1	0	1	0	0

Phenomenon	Number of Sentences	Bad Data	Correct Full Parses	Incorrect Full Parses	Fragmentary Parses
C_Diathesis-Active-Trivalent-Direct_object_predicate(NP)	1	0	1	0	0
C_Diathesis-Active-Trivalent-Direct_object_VP_bare_infinitive	1	0	1	0	0
C_Diathesis-Active-Trivalent-Direct_object_VP_ed	1	0	1	0	0
C_Diathesis-Active-Trivalent-Direct_object_VP_ing	1	0	1	0	0
C_Diathesis-Active-Trivalent-Direct_object_VP_to_infinitive	1	0	1	0	0
C_Diathesis-Active-Trivalent-Indirect_object_complementizer(that)_indicative	1	0	1	0	0
C_Diathesis-Active-Trivalent-Indirect_object_complementizer(WH)_indicative	1	0	1	0	0
C_Diathesis-Active-Trivalent-Indirect_object_direct_object	2	0	2	0	0
C_Diathesis-Active-Trivalent-Indirect_object_PP	1	0	1	0	0
C_Diathesis-Active-Trivalent-Indirect_object_prep_VP_ing	2	0	2	0	0
C_Diathesis-Active-Trivalent-Indirect_object_VP_to_infinitive	1	0	1	0	0
C_Diathesis-Active-Trivalent-Particle_PP	1	0	1	0	0
C_Diathesis-Active-Trivalent-PP_complementizer(that)_indicative	1	0	1	0	0
C_Diathesis-Active-Trivalent-PP_complementizer(WH)_indicative	1	0	1	0	0
C_Diathesis-Active-Trivalent-PP_PP	1	0	1	0	0
C_Diathesis-Active-Trivalent-PP_VP_to_infinitive	1	0	1	0	0
C_Diathesis-Middle-Divalent-Direct_object	1	0	1	0	0
C_Diathesis-Passive	3	0	3	0	0
C_Diathesis-Passive-Auxiliary(be)	2	0	2	0	0
C_Diathesis-Passive-Auxiliary(get)	2	0	2	0	0
C_Diathesis-Passive-Divalent-As_AP	2	0	2	0	0
C_Diathesis-Passive-Divalent-As_NP	2	0	2	0	0
C_Diathesis-Passive-Divalent-As_VP_ing	2	0	2	0	0
C_Diathesis-Passive-Divalent-Complementizer(that)_indicative	2	0	2	0	0
C_Diathesis-Passive-Divalent-Complementizer(WH)_indicative	2	0	2	0	0
C_Diathesis-Passive-Divalent-PP	2	0	2	0	0
C_Diathesis-Passive-Divalent-Predicate(NP)	2	0	2	0	0
C_Diathesis-Passive-Divalent-Prep	1	0	1	0	0
C_Diathesis-Passive-Divalent-Prep_VP_ing	2	0	2	0	0
C_Diathesis-Passive-Divalent-VP_ed	8	0	8	0	0
C_Diathesis-Passive-Divalent-VP_ed_direct_object	2	0	2	0	0
C_Diathesis-Passive-Divalent-VP_ing	2	0	2	0	0
C_Diathesis-Passive-Divalent-VP_to_infinitive	4	0	4	0	0
C_Diathesis-Passive-Equi-Obj_controlled	3	0	3	0	0
C_Diathesis-Passive-Monovalent-Subj(NP)	4	0	4	0	0
C_Diathesis-Passive-No_agent	8	0	8	0	0
C_Diathesis-Passive-Obl_agent	1	0	1	0	0
C_Diathesis-Passive-Raising-Subj_to_obj-VP_to_infinitive	6	0	6	0	0
C_Diathesis-Passive-Tetravalent-VP_ed_particle_PP	2	0	2	0	0
C_Diathesis-Passive-Tetravalent-VP_ed_PP_complementizer(that)_indicative	4	0	4	0	0
C_Diathesis-Passive-Tetravalent-VP_ed_PP_PP	2	0	2	0	0
C_Diathesis-Passive-Trivalent-Direct_object_complementizer(that)_indicative	2	0	2	0	0
C_Diathesis-Passive-Trivalent-Direct_object_PP	2	0	2	0	0

Phenomenon	Number of Sentences	Bad Data	Correct Full Parses	Incorrect Full Parses	Fragmentary Parses
C_Diathesis-Passive-Trivalent-PP_complementizer(WH)_indicative	4	0	4	0	0
C_Diathesis-Passive-Trivalent-PP_VP_to_infinitive	3	0	3	0	0
C_Diathesis-Passive-Trivalent-VP_ed_complementizer(that)_indicative	2	0	2	0	0
C_Diathesis-Passive-Trivalent-VP_ed_complementizer(that)_subjunctive/should/bare_infinitive	1	0	1	0	0
C_Diathesis-Passive-Trivalent-VP_ed_complementizer(that)_subjunctive/should_bare_infinitive	3	0	3	0	0
C_Diathesis-Passive-Trivalent-VP_ed_particle	6	0	6	0	0
C_Diathesis-Passive-Trivalent-VP_ed_PP	6	0	6	0	0
C_Negation-Tense_aspect_modality-BE_HAVE-Past_tense	7	0	2	3	2
C_Negation-Tense_aspect_modality-BE_HAVE-Present_tense	7	0	2	3	2
C_Negation-Tense_aspect_modality-CAN-Past_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-CAN-Present_tense	16	0	8	2	6
C_Negation-Tense_aspect_modality-Contractions-BE_HAVE-Past_tense	7	0	2	3	2
C_Negation-Tense_aspect_modality-Contractions-BE_HAVE-Present_tense	7	0	5	1	1
C_Negation-Tense_aspect_modality-Contractions-CAN-Past_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-Contractions-CAN-Present_tense	8	0	4	1	3
C_Negation-Tense_aspect_modality-Contractions-DARE-Present_tense	8	0	8	0	0
C_Negation-Tense_aspect_modality-Contractions-DO_aux-Past_tense	1	0	0	1	0
C_Negation-Tense_aspect_modality-Contractions-DO_aux-Present_tense	1	0	0	1	0
C_Negation-Tense_aspect_modality-Contractions-MAY-Past_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-Contractions-MAY-Present_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-Contractions-MUST-Present_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-Contractions-NEED-Present_tense	8	0	6	1	1
C_Negation-Tense_aspect_modality-Contractions-OUGHT-Present_tense	8	0	4	2	2
C_Negation-Tense_aspect_modality-Contractions-SHALL-Past_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-Contractions-SHALL-Present_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-Contractions-USED-Past_tense	24	0	16	0	8
C_Negation-Tense_aspect_modality-Contractions-WILL-Past_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-Contractions-WILL-Present_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-DARE-Past_tense	8	0	8	0	0
C_Negation-Tense_aspect_modality-DARE-Present_tense	8	0	6	1	1
C_Negation-Tense_aspect_modality-DO_aux-Past_tense	1	0	0	1	0
C_Negation-Tense_aspect_modality-DO_aux-Present_tense	1	0	0	1	0
C_Negation-Tense_aspect_modality-MAY-Past_tense	9	0	3	4	2
C_Negation-Tense_aspect_modality-MAY-Present_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-MUST-Present_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-NEED-Present_tense	8	0	6	1	1
C_Negation-Tense_aspect_modality-OUGHT-Present_tense	8	0	4	2	2
C_Negation-Tense_aspect_modality-SHALL-Past_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-SHALL-Present_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-USED-Past_tense	23	0	12	2	9
C_Negation-Tense_aspect_modality-WILL-Past_tense	8	0	2	4	2
C_Negation-Tense_aspect_modality-WILL-Present_tense	8	0	2	4	2
C_Parentheticals-Embedded_parentheses	1	0	1	0	0

Phenomenon	Number of Sentences	Bad Data	Correct Full Parses	Incorrect Full Parses	Fragmentary Parses
C_Tense-Aspect-Modality-BE_HAVE-Past_tense	8	0	2	4	2
C_Tense-Aspect-Modality-BE_HAVE-Present_tense	8	0	2	4	2
C_Tense-Aspect-Modality-CAN-Past_tense	8	0	2	4	2
C_Tense-Aspect-Modality-CAN-Present_tense	8	0	4	1	3
C_Tense-Aspect-Modality-Contractions-BE_HAVE-Past_tense	4	0	1	2	1
C_Tense-Aspect-Modality-Contractions-BE_HAVE-Present_tense	7	0	2	4	1
C_Tense-Aspect-Modality-Contractions-CAN-Past_tense	4	0	1	2	1
C_Tense-Aspect-Modality-Contractions-MAY-Past_tense	4	0	1	2	1
C_Tense-Aspect-Modality-Contractions-MAY-Present_tense	4	0	1	2	1
C_Tense-Aspect-Modality-Contractions-MUST-Present_tense	4	0	1	2	1
C_Tense-Aspect-Modality-Contractions-SHALL-Past_tense	4	0	1	2	1
C_Tense-Aspect-Modality-Contractions-WILL-Past_tense	12	0	3	6	3
C_Tense-Aspect-Modality-Contractions-WILL-Present_tense	8	0	2	4	2
C_Tense-Aspect-Modality-DO_aux-Past_tense	1	0	0	1	0
C_Tense-Aspect-Modality-DO_aux-Present_tense	1	0	0	1	0
C_Tense-Aspect-Modality-MAY-Past_tense	8	0	2	4	2
C_Tense-Aspect-Modality-MAY-Present_tense	8	0	2	4	2
C_Tense-Aspect-Modality-MUST-Present_tense	8	0	2	4	2
C_Tense-Aspect-Modality-OUGHT-Present_tense	8	0	4	2	2
C_Tense-Aspect-Modality-SHALL-Past_tense	8	0	2	4	2
C_Tense-Aspect-Modality-SHALL-Present_tense	8	0	2	4	2
C_Tense-Aspect-Modality-USED-Past_tense	8	0	4	2	2
C_Tense-Aspect-Modality-WILL-Past_tense	8	0	2	4	2
C_Tense-Aspect-Modality-WILL-Present_tense	8	0	2	4	2
NP_Coordination-Case_assignment-Object_position	46	0	0	46	0
NP_Coordination-Case_assignment-Subject_position	89	0	0	89	0
NP_Modification-Relative_clauses-Non_restrictive-Genitive-Collectives	6	0	5	0	1
NP_Modification-Relative_clauses-Non_restrictive-Genitive-Nonpersonal	3	0	2	0	1
NP_Modification-Relative_clauses-Non_restrictive-Genitive-Personal	3	0	2	0	1
NP_Modification-Relative_clauses-Non_restrictive-Object_extraction-Collectives	4	0	3	0	1
NP_Modification-Relative_clauses-Non_restrictive-Object_extraction-Nonpersonal	2	0	1	0	1
NP_Modification-Relative_clauses-Non_restrictive-Object_extraction-Personal	3	0	2	0	1
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction(stranded)-Collectives	4	0	3	0	1
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction(stranded)-Nonpersonal	2	0	1	0	1
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction(stranded)-Personal	3	0	2	0	1
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction-Collectives	53	0	52	0	1
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction-Nonpersonal	27	0	26	0	1
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction-Personal	27	0	26	0	1
NP_Modification-Relative_clauses-Non_restrictive-Subject_extraction-Collectives	3	0	3	0	0
NP_Modification-Relative_clauses-Non_restrictive-Subject_extraction-Nonpersonal	2	0	2	0	0
NP_Modification-Relative_clauses-Non_restrictive-Subject_extraction-Personal	2	0	2	0	0
NP_Parentheticals-Embedded_parentheses	1	0	1	0	0
NP_Parentheticals-Omitted_item	2	0	2	0	0

Phenomenon	*Number of Sentences*	*Bad Data*	*Correct Full Parses*	*Incorrect Full Parses*	*Fragmentary Parses*
NP_Parentheticals-Punctuation_interaction	1	0	1	0	0
NP_Parentheticals-Punctuation_markers	4	0	4	0	0
S_Parentheticals-Punctuation_interaction	3	0	3	0	0
S_Types-Questions-Wh-Complementiser_adv	4	0	0	4	0
S_Types-Questions-Wh-Iobj(DET_wh+NP)	3	0	0	3	0
S_Types-Questions-Wh-Iobj(PRON_wh)	2	0	0	2	0
S_Types-Questions-Wh-Mod(DET_wh+ADV)	1	0	1	0	0
S_Types-Questions-Wh-Obj(DET_wh+NP)	3	0	0	3	0
S_Types-Questions-Wh-Obj(PRON_wh)	3	0	0	3	0
S_Types-Questions-Wh-Pobj(DET_wh+NP)	3	0	3	0	0
S_Types-Questions-Wh-Pobj(PP+DET_wh+NP)	3	0	0	3	0
S_Types-Questions-Wh-Pobj(PP+PRON_wh)	3	0	0	3	0
S_Types-Questions-Wh-Pobj(PRON_wh)	3	0	2	0	1
S_Types-Questions-Wh-Pred(ADV_wh)	1	0	0	1	0
S_Types-Questions-Wh-Pred(DET_wh+ADJ)	1	0	1	0	0
S_Types-Questions-Wh-Pred(DET_wh+NP)	1	0	1	0	0
S_Types-Questions-Wh-Pred(PRON_wh)	1	0	0	1	0
S_Types-Questions-Wh-Subj(DET_wh+NP)	3	0	3	0	0
S_Types-Questions-Wh-Subj(PRON_wh)	2	0	2	0	0
S_Types-Questions-Y/N_questions-Inverted	13	0	1	12	0
S_Types-Questions-Y/N_questions-Negative-Inverted-Non-tagged	10	0	3	7	0
S_Types-Questions-Y/N_questions-Non_inverted-Non-tagged	14	0	14	0	0

Appendix C-1: DIPETT v3.0 Detailed Performance on the TSNLP tense-aspect-modality Phenomena

Appendix C-1 shows the specific performance of DIPETT v3.0 on the grammatical TSNLP sentences testing tenses in combination with aspects and modals.

✓ Indicates that a full parse tree was produced and the representation is correct.
✗ Indicates that a fragmentary parse or an incorrect parse tree was produced.

	Active							Passive							
	succeed	succeeded	succeeding	be succeeding	been succeeding	have succeeded	have been succeeding	seen	be seen	been seen	being seen	be being seen	been being seen	have been seen	have been being seen
(no modal)	✓	✓													
can	✓			✗		✗	✗		✗			✗		✗	✗
could	✓			✓		✓	✓		✗			✗		✗	✗
could've		✓			✓					✗			✗		
'd	✓	✓		✓	✓	✓	✓		✗	✗		✗	✗	✗	✗
did	✓														
does	✓														
had		✓			✓					✗			✗		
has		✓			✓					✗			✗		
is			✓					✗			✗				
'll	✓			✓		✓	✓		✗			✗		✗	✗
may	✓			✓		✓	✓		✗			✗		✗	✗
may've		✓			✓					✗			✗		
might	✓			✓		✓	✓		✗			✗		✗	✗
might've		✓			✓					✗			✗		
must	✓			✓		✓	✓		✗			✗		✗	✗
must've		✓			✓					✗			✗		
ought to	✓			✓		✗	✗		✗			✗		✗	✗
's		✓	✓		✓			✓		✗	✗		✗		
shall	✓			✓		✓	✓		✗			✗		✗	✗
should	✓			✓		✓	✓		✗			✗		✗	✗
should've		✓			✓					✗			✗		
used to	✓			✓		✗	✗		✗			✗		✗	✗
was			✓					✗			✗				
will	✓			✓		✓	✓		✗			✗		✗	✗
would	✓			✓		✓	✓		✗			✗		✗	✗
would've		✓			✓					✗			✗		

	succeed	succeeded	succeeding	be succeeding	been succeeding	have succeeded	have been succeeding	seen	be seen	been seen	being seen	be being seen	been being seen	have been seen	have been being seen
can not	✔			✘		✘	✘		✘			✘		✘	✘
cannot	✔			✘		✘	✘		✘			✘		✘	✘
can't	✔			✘		✘	✘		✘			✘		✘	✘
could not	✔			✔		✔	✔		✘			✘		✘	✘
couldn't	✔			✔		✔	✔		✘			✘		✘	✘
dare not	✔			✘		✘	✘		✘			✘		✘	✘
dared not	✘			✘		✘	✘		✘			✘		✘	✘
daren't	✘			✘		✘	✘		✘			✘		✘	✘
did not	✔														
did not use to	✘			✘		✘	✘					✘		✘	✘
did not used to	✘			✘		✘	✘					✘		✘	✘
didn't	✔														
didn't use to	✘			✘		✘	✘		✘			✘		✘	✘
didn't used to	✘			✘		✘	✘		✘			✘		✘	✘
does not	✔														
doesn't	✔														
had not		✔			✔					✘			✘		
hadn't		✔			✔					✘			✘		
has not		✔			✔					✘			✘		
hasn't		✘			✘					✘			✘		
is not			✔					✘			✘				
isn't			✔					✘			✘				
may not	✔			✔		✔	✔		✘			✘		✘	✘
mayn't	✔			✔		✔	✔		✘			✘		✘	✘
might not	✔			✔		✔	✔		✘			✘		✘	✘
mightn't	✔			✔		✔	✔		✘			✘		✘	✘
must not	✔			✔		✔	✔		✘			✘		✘	✘
mustn't	✔			✔		✔	✔		✘			✘		✘	✘
need not	✔			✘		✘	✘		✘			✘		✘	✘
needn't	✔			✘		✘	✘		✘			✘		✘	✘
not might							✔								
ought not to	✔			✔		✘	✘		✘			✘		✘	✘
oughtn't to	✔			✔		✘	✘		✘			✘		✘	✘
shall not	✔			✔		✔	✔		✘			✘		✘	✘
shan't	✔			✔		✔	✔		✘			✘		✘	✘
should not	✔			✔		✔	✔		✘			✘		✘	✘
shouldn't	✔			✔		✔	✔		✘			✘		✘	✘
used not to	✔			✔		✘	✘		✘			✘		✘	✘
usedn't to	✘			✘		✘	✘		✘			✘		✘	✘
was not			✔					✘			✘				
wasn't			✔					✘			✘				
will not	✔			✔		✔	✔		✘			✘		✘	✘
won't	✔			✔		✔	✔		✘			✘		✘	✘
would not	✔			✔		✔	✔		✘			✘		✘	✘
wouldn't	✔			✔		✔	✔		✘			✘		✘	✘

Appendix D: DEVAL — DIPETT Progress Evaluation Program

```
/****************************************************************************/
/* file: deval.pl                                                           */
/* version: April, 1999                                                     */
/* author: E. Scarlett                                                      */
/****************************************************************************/

:- dynamic outstruct/4, dict/4, idiom/4.
:- multifile dict/4, idiom/4.

:- ensure_loaded( [ library(files),
                    library(directory),
                    library(ctypes),
                    library(basics),
                    library(date),
                    library(lists) ] ) ;
   ensure_loaded( ['COMPAT_LIB/files',
                   'COMPAT_LIB/directory',
                   'COMPAT_LIB/ctypes',
                   'COMPAT_LIB/basics',
                   'COMPAT_LIB/date',
                   'COMPAT_LIB/lists' ] ).

check_dipett_loaded :-
        clause(user:parser_copyright(_),_), !.

load_rw :- check_dipett_loaded.
load_rw :- ['dipett_1_rw.pl'].

load_bootdipett :- check_dipett_loaded,  ['deval_bootdipett.pl'].
load_bootdipett.

:- ['deval_driver.pl'].
:- load_rw.
:- load_bootdipett.
:- init_test_suite_dir.
:- init_suite_list.


test :-
        menu(0).

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

```
%
%               K N O W N   P A R S E   T R E E   T Y P E S
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

parse_success(parse_tree__decla_or_imper).
parse_success(parse_tree__inter).
parse_success(parse_tree__inter_decla_or_imper).

parse_failure(parse_tree__frag_series).
parse_failure(parse_tree__frag).
parse_failure([]).
parse_failure(error).
parse_failure(X) :- parse_success(X), !, fail.
parse_failure(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%               C O U N T E R S
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
% I N I T   A N D   R E S E T   C O U N T E R S
%

zero_counters :-
        retractall(counter(_, _)),
        assertz(counter(diff_sent, 0)),
        assertz(counter(diff_trees, 0)).

destroy_counter(CounterName) :-
        retractall(counter(CounterName, _)).

%
% I N C R E M E N T   C O U N T E R
%

inc_counter(CounterName) :-
        counter(CounterName, N),
        retractall(counter(CounterName, _)),
        N1 is N + 1,
        assertz(counter(CounterName, N1)).
inc_counter(CounterName) :-
        assertz(counter(CounterName, 1)).

%
% R E P O R T I N G   C O U N T E R   V A L U E S
```

```
%

% Name is/has one thing
write_counter(CounterName, Is, Unit, _MakeItPlural) :-
        counter(CounterName, 1),
        msg([CounterName, ' ', Is, ' 1 ', Unit, nl]).
% Name is/has N things
write_counter(CounterName, Is, Unit, MakeItPlural) :-
        counter(CounterName, N),
        msg([CounterName, ' ', Is, ' ', N, ' ', Unit, MakeItPlural, nl]).
% undefined counter means it is 0
write_counter(CounterName, Is, Unit, MakeItPlural) :-
        msg([CounterName, ' ', Is, ' 0 ', Unit, MakeItPlural, nl]).

% write "one" thing
write_counter_msg(Msg, CounterName, Unit, _MakeItPlural) :-
        counter(CounterName, 1),
        msg([Msg, ' 1 ', Unit, nl]).
% N things
write_counter_msg(Msg, CounterName, Unit, MakeItPlural) :-
        counter(CounterName, N),
        msg([Msg, ' ', N, ' ', Unit, MakeItPlural, nl]).
% undefined means 0 things
write_counter_msg(Msg, _CounterName, Unit, MakeItPlural) :-
        msg([Msg, ' 0 ', Unit, MakeItPlural, nl]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%     R E P O R T I N G   P A R S E   T R E E   D I F F E R E N C E S
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/* report number of sentences and parse trees that are different */
report_differences :-
        counter(diff_sent, S), counter(diff_trees, T),
        msg([nl, 'Summary of parse tree comparisons:']),
        cheer(S, T),
        reportd(S, sentence), reportd(S, remainder), reportd(T, 'parse tree'),
        nld, nld, !.
report_differences :-
        msg([deval, 'Oops, failed to report', nl, nl]), !.

cheer(0, 0) :- msg([nl, ' YES!!!']).
cheer(_, _) :- msg([nl, ' Hmm...']).

reportd(0, remainder).
reportd(_, remainder) :-
        msg([nl, ' Of the remainder --']).
```

```
reportd(0, What) :-
	msg([nl, ' The ', What, 's are all the same.']).
reportd(1, What) :-
	msg([nl, ' One ', What, ' is not the same.']).
reportd(N, What) :-
	msg([nl, ' ', N, ' ', What, 's are not the same.']).

/* report tree types and how many of each type */
report_tree_types(Label) :-
	counter([Label, TreeType], C),
	write_type(C, TreeType).
report_tree_types(_) :- nld.

write_type(C, TreeType) :-
	((C < 10, writed(' ')) ; C >= 10),
	((C < 100, writed(' ')) ; C >= 100),
	((C < 1000, writed(' ')) ; C >= 1000),
	writed(C), writed('  '), writed(TreeType), nld, fail.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%		P A R S I N G   A N D   C O M P A R I N G   T R E E S
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

parse_and_compare(Suite) :-
	suite_files(Suite, Sent, Dict, DictAdd, OutStructs, ScriptFile, BibleTrees),
	open_script_file(ScriptFile),
	msg([nl, deval, 'Beginning parse session on ', Suite]),
	write_date_and_time, nld,
	write_cpu_time_limit,
	run_dipett_session(Suite, Sent, Dict, DictAdd, OutStructs),
	msg([nl, deval, 'Comparing parse results to bible trees for ',Suite,' sentences', nl]),
	compare_trees(OutStructs, parse_tree, BibleTrees, bible_tree),
	close_script_file(ScriptFile).
%	report_stats.
parse_and_compare(Suite) :-
	msg([nl, deval, 'Failed parse_and_compare on ', Suite, nl]).
%	report_stats.

compare_no_parse(Suite) :-
	get_out_file_name(Suite, ParseTreeFile),
	get_bible_file_name(Suite, BibleTreeFile),
	msg([nl, deval, 'Comparing parse results to bible trees for ',Suite,' sentences', nl]),
	compare_trees(ParseTreeFile, parse_tree, BibleTreeFile, bible_tree).
compare_no_parse(Suite) :-
	msg([nl, deval, 'Failed compare_no_parse on ', Suite, nl]).
```

```
compare_OutFiles(Suite1, Suite2) :-
        get_out_file_name(Suite1, File1),
        get_out_file_name(Suite2, File2),
        msg([nl, deval, 'Comparing parse results for ',Suite1,' sentences to ',Suite2,' sentences',nl]),
        compare_trees(File1, Suite1, File2, Suite2).
compare_OutFiles(_, _) :-
        msg([deval,'Failed compare_OutFiles',nl]).

compare_trees(File1, Label1, File2, Label2) :-
        zero_counters,
        load_trees(File1, Label1),
        load_trees(File2, Label2), !,

        msg([nl, 'Report:',nl, nl]),

        write_counter(Label1, has, unit, s),
        report_tree_types(Label1),
        write_counter(Label2, has, unit, s),
        report_tree_types(Label2),

        msg([nl, 'Sentences where neither ', Label1, ' nor ', Label2, ' parsed successfully',nl]),
        destroy_counter(ok_not),
        list_success_failure2(1, Label1, parse_fail, Label2, parse_fail),
        write_counter_msg('  ', ok_not, 'sentence', 's'),

        msg([nl, 'Sentences where ',  Label1, ' has a successful parse but ', Label2, ' does not:', nl]),
        destroy_counter(ok_not),
        list_success_failure2(1, Label1, parse_ok, Label2, parse_fail),
        write_counter_msg('  ', ok_not, 'sentence', 's'),

        msg([nl, 'Sentences where ', Label2, ' has a successful parse but ', Label1, ' does not:', nl]),
        destroy_counter(ok_not),
        list_success_failure2(1, Label1, parse_fail, Label2, parse_ok),
        write_counter_msg('  ', ok_not, 'sentence', 's'),

        msg([nl, 'Sentences that parsed successfully and have identical parse trees:', nl]),
        destroy_counter(ok_not),
        list_success_failure2(1, Label1, parse_ok, Label2, parse_same),
        write_counter_msg('  ', ok_not, 'sentence', 's'),

        msg([nl, 'Comparision of full parse trees that are not identical:', nl]),
        destroy_counter(ok_not),
        compare_all_trees(1, Label1, Label2),
        write_counter_msg('  ', ok_not, 'sentence', 's'),
        report_differences,

        destroy_counter(ok_not),
```

```
        unload_trees(File1, Label1),
        unload_trees(File2, Label2).

% this is in case Label1 trees loaded successfully but Label2 did not
compare_trees(File1, Label1, File2, Label2) :-
        outstruct(Label1, _Unit, _Sentence, _TreeStruct),
        msg([nl, 'Report:',nl, nl]),
        write_counter(Label1, has, unit, s),
        report_tree_types(Label1),

        msg([nl, Label1, ' sentences that parsed successfully:', nl]),
        destroy_counter(ok_not),
        list_success_failure1(1, Label1, parse_ok),
        write_counter_msg('  ', ok_not, 'sentence', 's'), nld,

        msg([nl, Label1, ' sentences that failed to parse:', nl]),
        destroy_counter(ok_not),
        list_success_failure1(1, Label1, parse_fail),
        write_counter_msg('  ', ok_not, 'sentence', 's'), nld,

        unload_trees(File1, Label1),
        unload_trees(File2, Label2).

compare_trees(File1, Label1, File2, Label2) :-
        msg([deval, 'Comparison stopped', nl]),
        unload_trees(File1, Label1),
        unload_trees(File2, Label2).


%
%     L I S T   S U C C E S S   F A I L U R E  2
%

/* partitions trees from two files according to the specified conditions:
** both-failed, both-succeeded, one-failed, the-other-failed,
** and lists all the sentences that meet the specified condition
*/
list_success_failure2(Unit, Label1, Condition1, Label2, Condition2) :-
        outstruct(Label1, Unit, Sentence1, TreeStruct1),
        outstruct(Label2, Unit, Sentence2, TreeStruct2),
        sift_ok_not_ok(Unit, Label1, Condition1, Sentence1, TreeStruct1,
                             Label2, Condition2, Sentence2, TreeStruct2),
        NextUnit is Unit + 1,
        list_success_failure2(NextUnit, Label1, Condition1, Label2, Condition2).
list_success_failure2(_Unit, _Label1, _Condition1, _Label2, _Condition2).


/* sentences are the same, parse trees are the same, report sentence if a successful parse */
```

```
sift_ok_not_ok(Unit, _Label1, parse_ok, Sentence, TreeStruct,
                     _Label2, parse_same, Sentence, TreeStruct) :-
        TreeStruct =.. [TreeType | _RestTree ],
       report_ok_not_ok(Unit, Sentence, parse_ok, TreeType, parse_ok, TreeType).

/* sentences are the same, report sentence if tree types match conditions */
sift_ok_not_ok(Unit, _Label1, Condition1, Sentence, TreeStruct1,
                     _Label2, Condition2, Sentence, TreeStruct2) :-
        TreeStruct1 =.. [TreeType1 | _RestTree1 ],
        TreeStruct2 =.. [TreeType2 | _RestTree2 ],
       report_ok_not_ok(Unit, Sentence, Condition1, TreeType1, Condition2, TreeType2).

/* sentences are different -- no report necessary */
sift_ok_not_ok(_Unit, _Label1, _Condition1, _Sentence1, _TreeStruct1,
                      _Label2, _Condition2, _Sentence2, _TreeStruct2).


/* If tree types match the specified conditions, then report.
** Looking for trees that are both successful parses.
*/
report_ok_not_ok(Unit, Sentence, parse_ok, TreeType1, parse_ok, TreeType2) :-
       parse_success(TreeType1),
       parse_success(TreeType2), !,
       inc_counter(ok_not),
       msg([' Unit #', Unit, ':  ', Sentence, nl]).

/* looking for tree1 is ok, tree2 failed */
report_ok_not_ok(Unit, Sentence, parse_ok, TreeType1, parse_fail, TreeType2) :-
       parse_success(TreeType1),
       parse_failure(TreeType2), !,
       inc_counter(ok_not),
       msg([' Unit #', Unit, ':  ', Sentence, nl]).

/* looking for tree1 failed, tree2 is ok */
report_ok_not_ok(Unit, Sentence, parse_fail, TreeType1, parse_ok, TreeType2) :-
       parse_failure(TreeType1),
       parse_success(TreeType2), !,
       inc_counter(ok_not),
       msg([' Unit #', Unit, ':  ', Sentence, nl]).

/* looking for both-failed */
report_ok_not_ok(Unit, Sentence, parse_fail, TreeType1, parse_fail, TreeType2) :-
       parse_failure(TreeType1),
       parse_failure(TreeType2),
       inc_counter(ok_not), !,
       msg([' Unit #', Unit, ':  ', Sentence, nl]).

/* succeed no matter what */
```

```
report_ok_not_ok(_Unit, _Sentence, _Condition1, _TreeType1, _Condition2, _TreeType2).


%
%        L I S T   S U C C E S S   F A I L U R E  1
%

/* partitions a single file into parse successes and parse failures,
** and writes the sentences that match the specified condition
*/
list_success_failure1(Unit, Label, Condition) :-
        outstruct(Label, Unit, Sentence, TreeStruct),
        TreeStruct =.. [TreeType | _RestTree ],
        report_ok_not_ok_single(Unit, Sentence, TreeType, Condition),
        NextUnit is Unit + 1,
        list_success_failure1(NextUnit, Label).
list_success_failure1(_Unit, _Label).

/* looking for parse successes, report sentences that parsed successfully*/
report_ok_not_ok_single(Unit, Sentence, TreeType, parse_success) :-
        parse_success(TreeType),
        inc_counter(ok_not),
        msg([' Unit #', Unit, ':  ', Sentence, nl]).

/* looking for parse failures, report sentences that failed to parse*/
report_ok_not_ok_single(Unit, Sentence, TreeType, parse_fail) :-
        parse_failure(TreeType),
        inc_counter(ok_not),
        msg([' Unit #', Unit, ':  ', Sentence, nl]).

/* succeed no matter what */
report_ok_not_ok_single(_Unit, _Sentence, _TreeType, _Condition).


%
%        C O M P A R E   A L L   T R E E S
%

% get specified parse_tree and bible_tree, compare them, write results,
% and start over again with the next set
compare_all_trees(Unit, Label1, Label2) :-
        outstruct(Label1, Unit, Sentence1, TreeStruct1),
        outstruct(Label2, Unit, Sentence2, TreeStruct2), !,
        compare_parse_tree_structs(Unit, Label1, Sentence1, TreeStruct1, Label2, Sentence2, TreeStruct2),
        NextUnit is Unit + 1,
        compare_all_trees(NextUnit, Label1, Label2).

% one is missing (Label2 outstruct) but the other is there (Label1)
```

```
compare_all_trees(Unit, Label1, Label2) :-
        outstruct(Label1, Unit, _Sentence, _TreeStruct), !,
        msg([' Unit #', Unit, nl,
             '  ', Label2, ' has no outstruct loaded for unit ', Unit, nl, nl]),
        NextUnit is Unit + 1,
        compare_all_trees(NextUnit, Label1, Label2).

% one is missing (Label1 outstruct) but the other is there (Label2)
compare_all_trees(Unit, Label1, Label2) :-
        outstruct(Label2, Unit, _Sentence, _TreeStruct), !,
        msg([' Unit #', Unit, nl,
             '  ', Label1, ' has no outstruct loaded for unit ', Unit, nl, nl]),
        NextUnit is Unit + 1,
        compare_all_trees(NextUnit, Label1, Label2).

% processing stops when both parse_tree units are not found
compare_all_trees(_Unit, _Label1, _Label2).

%
%       C O M P A R E   P A R S E   T R E E   S T R U C T S
%

% both the sentences and the trees are the same so no comparison is necessary
compare_parse_tree_structs(_Unit, _Label1, Sentence, TreeStruct, _Label2, Sentence, TreeStruct) :- !.

% the sentences are the same, but the trees are different: compare only if both are full parse trees
compare_parse_tree_structs(Unit, Label1, Sentence, TreeStruct1, Label2, Sentence, TreeStruct2) :-
         TreeStruct1 =.. [TreeType1 | _RestTree1],
         TreeStruct2 =.. [TreeType2 | _RestTree2],
        parse_success(TreeType1), parse_success(TreeType2),
        inc_counter(ok_not),
        inc_counter(diff_trees),
        msg([' Unit #', Unit, ':  ', Sentence, nl]),
        show_difference(Label1, TreeStruct1, Label2, TreeStruct2), !.

% the sentences are the same, the parse trees are different, but at least one parse failed
% so don't bother showing it
compare_parse_tree_structs(_Unit, _Label1, Sentence, _TreeStruct1, _Label2, Sentence, _TreeStruct2) :- !.

% the sentences are diffent, never mind the trees
compare_parse_tree_structs(Unit, Label1, Sentence1, _TreeStruct1, Label2, Sentence2, _TreeStruct2) :-
        inc_counter(diff_sent),
        inc_counter(ok_not),
        msg([' Unit #', Unit, ':  different sentences', nl, '  ',
             Label1, ' has: ', Sentence1, nl, '  ',
             Label2, ' has: ', Sentence2, nl, nl]), !.

%
```

```
%       S H O W   D I F F E R E N C E
%

% two items the same obviously aren't different so just return
% (this shouldn't happen)
show_difference(_Label1, TreeStruct, _Label2, TreeStruct).

% two different atoms => write them out and we're done
show_difference(Label1, TreeStruct1, Label2, TreeStruct2) :-
        atom(TreeStruct1), atom(TreeStruct2),
        msg(['  ', Label1, ' has: ', TreeStruct1, nl,
             '  ', Label2, ' has: ', TreeStruct2, nl, nl]).

% two different lists but the head is the same => work on the tails
show_difference(Label1, [H|T1], Label2, [H|T2]) :-
        show_difference(Label1, T1, Label2, T2).

% two different lists and the head is different => look closer at the heads
% - we're just showing the first difference we find, so the tails don't
% matter when the heads are different
show_difference(Label1, [H1|_], Label2, [H2|_]) :-
        show_difference(Label1, H1, Label2, H2).

% could be two different anything, even different functor or arity, so
% process whatever-it-is by converting it to a list and looking at each
% element of that
show_difference(Label1, Struct1, Label2, Struct2) :-
        Struct1 =.. List1, Struct2 =.. List2,
        show_difference(Label1, List1, Label2, List2).

% we should never get here, but just in case, succeed anyway
show_difference(Label1, Struct1, Label2, Struct2) :-
        msg([deval, 'Error - show_difference failed', nl,
            '  ', Label1, ' has: ', Struct1, nl,
            '  ', Label2, ' has: ', Struct2, nl, nl]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                L O A D I N G   P A R S E   T R E E S
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
% load a file of DIPETT parse trees, assert each as Pred(N,S,T).
%
load_trees(TreeFile, Label) :-
        msg([deval, 'Loading ', Label, ' from file ', TreeFile, nl]),
        file_exists(TreeFile), !,
```

```
        unload_trees(quiet, Label),
        see(TreeFile),
        assert_trees(Label),
        seen, !.
load_trees(TreeFile, _) :-
        msg([deval, 'File does not exist - ', TreeFile, nl]),
        fail, !.

assert_trees(Label) :-
        read(Tree), process(Tree, Label).
assert_trees(_).

process(end_of_file, _) :- !.
process(Tree, Label) :-
        Tree =.. [parse_tree, Unit, Sentence, ParseStruct],
        NewTree =.. [outstruct, Label, Unit, Sentence, ParseStruct],
        inc_counter(Label),
        count_tree_type(Label, ParseStruct),
        assertz(NewTree),
        assert_trees(Label).

unload_trees(quiet, Label) :-
        Trees =.. [outstruct, Label, _, _, _],
        retractall(Trees),
        destroy_counter(Label),
        destroy_counter([Label, _]).
unload_trees(TreeFile, Label) :-
        msg([deval,'Unloading ', Label, ' file ', TreeFile, nl]),
        unload_trees(quiet, Label).

count_tree_type(Label, ParseStruct) :-
         ParseStruct =.. [TreeType | _RestTree ],
        inc_counter([Label, TreeType]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%            U N L O A D   A U X   D I C T I O N A R Y
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

unload_aux_dict(DictFile) :-
        file_must_exist(DictFile), !,
        msg([deval, 'Unloading auxiliary dictionary - ', DictFile, nl]),
        see(DictFile),
        unload_dict_entries,
        seen.

unload_dict_entries :-
```

```
        read(DictEntry),
        unload_dict_entry(DictEntry).
unload_dict_entries.

unload_dict_entry(end_of_file) :- !.
unload_dict_entry(Entry) :-
        check_entry(Entry),
        retract(Entry),
        unload_dict_entries.
unload_dict_entry(_) :-
        unload_dict_entries.

check_entry(dict(A,B,C,D)) :- dict(A,B,C,D).
check_entry(idiom(A,B,C,D)) :- idiom(A,B,C,D).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%            R E A D I N G   &   W R I T I N G
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
% ask the user a question, reply with the answer
%
ask_user(Prompt, Answer) :-
        write(Prompt), writed(' '),
        read_line(Line), name(Answer, Line), nld, !.

ask_user(Prompt, Answer, Default) :-
        write(Prompt), writed(' ['), writed(Default), writed('] '),
        read_line(Line), form_answer(Line, Default, Answer), nld, !.

form_answer([], Default, Default).
form_answer(Line, _, Answer) :- name(Answer, Line).


open_report_file :- want_to_save9(defaults).

want_to_save9(_) :-
            clause(user:report_filename(_FN), true),
            writep('d-1) Change the current DEVAL report filename [y/N]? '),
            read_answer(Answer),
            ((Answer == y, close_report_file,
                fail) ;
             Answer \== y).
want_to_save9(Defaults) :-
            check_report_filename_defaults(Defaults, Fname), !,
```

```
            repeat,
            writep('D-1) DEVAL report filename '),
            writep('["name"/'), writep(Fname), writep(']? '),
            read_answer(FileName9),
            check_report_filename(FileName9, FileName9_ok),
            new_file_ok(Defaults, FileName9_ok),
        retractall(report_file(_)),
        retractall(report_filename(_)),
        open(FileName9_ok, write, Stream9),
        write_date_and_time(Stream9, FileName9_ok),
        assert(report_file(Stream9)),
        assert(report_filename(FileName9_ok)).

close_report_file  :-
        clause(user:report_file(Stream9), true),
            clause(user:report_filename(FN), true),
        writed('*deval*  Closing report file - '), writed(FN), nld,
        nl(Stream9), write_date_and_time(Stream9, FN),
        flush_output(Stream9),
        close(Stream9),
        retractall(report_file(_)),
        retractall(report_filename(_)).
close_report_file(_).    % it doesn't exist, succeed.

check_report_filename(F, F)     :- name(F,[Char|_]), is_alpha(Char), !.
check_report_filename(_, Fname) :-
    check_report_filename_defaults(defaults, Fname).

check_report_filename_defaults(defaults,'zzz.deval-report.script').


nld :-  clause(user:report_file(Stream9),true), nl(Stream9),
        flush_output(Stream9), nlp, !.
nld :-        nlp.

writed(X) :- clause(user:report_file(Stream9),true), write(Stream9, X),
             writep(X), !.
writed(X) :- writep(X).

msg([]) :- !.
msg([nl|RestMsg]) :- nld, msg(RestMsg), !.
msg([nld|RestMsg]) :- nld, msg(RestMsg), !.
msg([deval|RestMsg]) :- writed('*deval*  '), msg(RestMsg), !.
msg([Item|RestMsg]) :- writed(Item), msg(RestMsg), !.

%
%       R E P O R T   S T A T S
%
```

```
report_stats :-
        statistics(memory, [MemUsed, MemFree]),
        statistics(program, [ProgSpace, _]),
        statistics(atoms, [Natoms, AtomSpaceUsed, AtomSpaceFree]),

        statistics(stacks, [TotalGlobal, TotalLocal]),
        statistics(global_stack, [GlobalStackUsed, GlobalStackFree]),
        statistics(local_stack, [LocalStackUsed, LocalStackFree]),

        statistics(garbage_collection, [Ngc, FreedBytes, GCTime]),
        statistics(stack_shifts, [GlobalStackShifts, LocalStackShifts, SSTime]),
        statistics(atom_garbage_collection, [Nagc, AtomSpaceFreed, AGCTime]),

        nld,
        outtotal('memory (total) ', 'bytes', MemUsed, MemFree),
        outbytes('program space  ', ProgSpace),
        outriple('global space   ', 'bytes', TotalGlobal, GlobalStackUsed, GlobalStackFree),
        outriple('local space    ', 'bytes', TotalLocal, LocalStackUsed, LocalStackFree),
        outriple('atom space     ', 'atoms', Natoms, AtomSpaceUsed, AtomSpaceFree),
        nld,
        outtime(SSTime, GlobalStackShifts, 'global and', LocalStackShifts, 'local space shifts'),
        outtime(GCTime, Ngc, 'garbage collections which collected', FreedBytes, 'bytes'),
        outtime(AGCTime, Nagc, 'atom garbage collections which collected', AtomSpaceFreed, 'bytes'),
        nld, !.

report_stats :- !.

outbytes(Stat, Bytes) :-
        msg([Stat, Bytes, ' bytes', nl]).

outtotal(Stat, Unit, Used, Free) :-
        Total is Used + Free,
        outriple(Stat, Unit, Total, Used, Free).

outriple(Stat, Unit, Total, Used, Free) :-
        msg([Stat, Total, ' ', Unit, ' : ', Used, ' in use + ', Free, ' free', nl]).

outtime(Time, N1, Msg1, N2, Msg2) :-
        msg([Time, ' msec for ', N1, ' ', Msg1, ' ', N2, ' ', Msg2, nl]).



/*****************************************************************************/
/* file: deval_driver.pl                                                     */
/* version: April, 1999                                                      */
/* author: E. Scarlett                                                       */
/*****************************************************************************/
```

```
:- dynamic test_suite_dir/1.

menu(0) :-
        nl, nl, write('** deval ** Main menu'), nl,
        write('1. Run DIPETT and then compare the output trees to bible trees'), nl,
        write('2. Compare DIPETT output to bible trees'), nl,
        write('3. Compare parse results for two different DIPETT output files'), nl,
        write('4. Change the default directory'), nl,
        write('h. Help'), nl,
        write('q. Quit'), nl,
        nl, ask_user('What do you want to do?', Answer),
        process_menu_item(0, Answer).

menu(N) :-
        member(N, [1,2,3]),
        banner(N, Banner),
        nl, nl, write(Banner), nl,
        write_dir_choices,
        write('m. Go back to the main menu'), nl,
        write('h. Help'), nl,
        write('q. Quit'), nl,
        nl, get_prompt(N, Prompt), ask_user(Prompt, Answer),
        process_menu_item(N, Answer).

get_prompt(3, 'Choose two directories (e.g. 1,2) : ').
get_prompt(_, 'Which directories? ').

banner(1, '** deval ** Run DIPETT on which sentences?').
banner(2, '** deval ** Compare parse trees and bible trees for which directory?').
banner(3, '** deval ** Compare parse trees for which directory pair?').

write_dir_choices :-
        suite(N, S),
        write(N), write('. '), write(S), nl, fail.
write_dir_choices.


% Go back to the main menu
process_menu_item(_, m) :- menu(0).

% Write some help
process_menu_item(0, h) :-
        write('Enter a number, e.g. "3"'), nl, menu(0).
process_menu_item(3, h) :-
        write('Enter a pair of numbers separated by a comma, e.g. "1,2"'), nl, menu(3).
process_menu_item(N, h) :-
        write('Enter a number or set separated by commas, e.g. "3" or "2,3,5"'), nl, menu(N).
```

```
% Stop now.
process_menu_item(_, x) :- halt.        % exit prolog to unix prompt
process_menu_item(_, e) :- halt.
process_menu_item(_, a).                 % abort to prolog prompt
process_menu_item(_, q).

%
%         M E N U ( 0 )  -  M A I N   M E N U
%
process_menu_item(0, 1) :-
        check_dipett_loaded,
        menu(1).
process_menu_item(0, 1) :-
        \+ check_dipett_loaded,
        nl, write('*deval*  DIPETT is not loaded.  Load deval into dhq and try again...'), nl,
        menu(0).
process_menu_item(0, 2) :- menu(2).
process_menu_item(0, 3) :- menu(3).
process_menu_item(0, _) :- menu(0).


%
%         M E N U ( 1 ) - R U N   D I P E T T  &  C O M P A R E
%

% one specific directory
process_menu_item(1, N) :-
        number(N),
        suite(N, Suite), !,
        input_files_exist(parse, Suite), !,
        write('*deval*  Run DIPETT and compare parse results on sentences'), nl,
        parse_and_compare_one(Suite).

% a list of directories
process_menu_item(1, Input) :-
        atom(Input), !,
        cvt_input_to_list(parse, Input, DirList), !,
        write('*deval*  Run DIPETT and compare parse results on sentences'), nl,
        parse_and_compare_list(DirList).

% handle failure graciously
process_menu_item(1, N) :- suite(N, _), menu(1).


%
%         M E N U ( 2 ) -  C O M P A R E   P A R S E   T R E E S   T O   B I B L E   T R E E S
%
```

```
% one specific directory
process_menu_item(2, N) :-
        number(N),
        suite(N, Suite), !,
        input_files_exist(compare_bible, Suite), !,
        write('*deval*  Compare parse trees to bible trees'), nl,
        compare_one(Suite).

% a list of directories
process_menu_item(2, Input) :-
        atom(Input), !,
        cvt_input_to_list(compare_bible, Input, DirList), !,
        write('*deval*  Compare parse trees to bible trees'), nl,
        compare_list(DirList).

% handle failure graciously
process_menu_item(2, N) :- suite(N, _), menu(2).


%
%         M E N U ( 3 ) -  C O M P A R E   P A R S E   T R E E S   T O   P A R S E   T R E E S
%

% one specific directory
process_menu_item(3, N) :-
        number(N), write('Invalid input, try again.'), nl, nl, menu(3).

% a list of directories -  valid input is a directory pair
process_menu_item(3, Input) :-
        atom(Input),
        cvt_input_to_list(compare_tree, Input, DirList),
        write('*deval*  Compare parse trees'), nl,
        compare_parse_tree_pair(DirList).

% handle failure graciously
process_menu_item(3, N) :- suite(N, _), menu(3).


%
%         M E N U ( 4 ) -  C H A N G E   D E F A U L T   D I R E C T O R Y
%

% one specific directory
process_menu_item(0, 4) :- change_test_suite_dir, menu(0).

% handle failure graciously
process_menu_item(0, 4) :- menu(0).
```

```
%
%        M E N U ( N , _ )  -  P R O C E S S   U N K N O W N   I N P U T
%

% Last chance -- Invalid or unknown input
process_menu_item(N, _) :-
        write('Invalid input, try again.'), nl, nl,
        menu(N).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%              P R O C E S S   I N P U T
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%       P A R S E   A N D   C O M P A R E   O N E
%
parse_and_compare_one(Suite) :-
        boot_dipett, !,
        zero_counters,
        open_report_file,
        parse_and_compare(Suite),
        close_report_file,
        byebye,
        halt.

%
%       C O M P A R E   O N E   ( N O   P A R S E )
%
compare_one(Suite) :-
        zero_counters,
        open_report_file,
        compare_no_parse(Suite),
        close_report_file.

%
%       P A R S E   A N D   C O M P A R E   L I S T
%
parse_and_compare_list(DirectoryList) :-
        boot_dipett, !,
        zero_counters,
        open_report_file,
        parse_and_compare_dir_list(DirectoryList),
```

```
        close_report_file,
        byebye,
        halt.

%
%       C O M P A R E   D I R   L I S T   ( N O   P A R S E )
%
compare_list(DirectoryList) :-
        zero_counters,
        open_report_file,
        compare_dir_list(DirectoryList),
        close_report_file.

%
%       C O M P A R E   P A R S E   T R E E   P A I R
%
compare_parse_tree_pair([Dir1, Dir2]) :-
        zero_counters,
        suite(Dir1, Suite1),
        suite(Dir2, Suite2),
        open_report_file,
        compare_OutFiles(Suite1, Suite2),
        close_report_file.

%
%        M A K E   C H A P T E R   L I S T   F R O M   I N P U T   S T R I N G
%

% take a user input string and make it into a list of directory numbers
% e.g.   user types in '1,2,3'   we return list of numbers   [1, 2, 3]
cvt_input_to_list(Files, Input, DirList) :-
        name(Input, InputList),
        make_number_list(InputList, [], DirList), !,
        dir_files_ok(Files, DirList).

% process input ascii, convert to a list of numbers,
% commas (44) separate numbers, ignore spaces (32),
% and fail if invalid input is encountered

make_number_list([], [], []).
% at the end of input, the current digit list is the last number in the output
make_number_list([], DigitList, [N]) :-
        name(N, DigitList), !,
        number(N).
% ignore spaces and continue
make_number_list([32|InList], DigitList, OutList) :-
        make_number_list(InList, DigitList, OutList).
% comma, convert digits to a number and append to final output
```

```
make_number_list([44|InList], DigitList, NewOutList) :-
        make_number_list(InList, [], OutList),
        name(N, DigitList), !,
        number(N),
        append([N], OutList, NewOutList).
% new digit, append to current digit list and continue processing
make_number_list([InChar|InList], DigitList, OutList) :-
        append(DigitList, [InChar], NewDigitList),
        make_number_list(InList, NewDigitList, OutList).

% check input files exist for each dir in a directory list
dir_files_ok(_Files, []).
dir_files_ok(Files, [Dir|DirList]) :-
        suite(Dir, Suite), !,
        input_files_exist(Files, Suite),
        dir_files_ok(Files, DirList).

% parse_and_compare a directory list constructed by cvt_input_to_list
parse_and_compare_dir_list([]).
parse_and_compare_dir_list([Dir|DirList]) :-
        suite(Dir, Suite),
        parse_and_compare(Suite),
        parse_and_compare_dir_list(DirList).
%if the head of the list failed, process the tail anyway
parse_and_compare_dir_list([_|DirList]) :-
        parse_and_compare_dir_list(DirList).

% compare (no parse) a directory list constructed by cvt_input_to_list
compare_dir_list([]).
compare_dir_list([Dir|DirList]) :-
        suite(Dir, Suite),
        compare_no_parse(Suite),
        compare_dir_list(DirList).
%if the head of the list failed, process the tail anyway
compare_dir_list([_|DirList]) :-
        compare_dir_list(DirList).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%               T E S T   S U I T E   D I R E C T O R Y   P R O C E S S I N G
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%        S U I T E  &  D I R E C T O R Y   N A M E S
%
```

```
init_test_suite_dir :-
        directory_exists('tests/tsnlp'),
        retractall(test_suite_dir(_)),
        assert(test_suite_dir('tests/')).
init_test_suite_dir :-
        directory_exists('../tsnlp'),
        retractall(test_suite_dir(_)),
        assert(test_suite_dir('../')).
init_test_suite_dir :-
        retractall(test_suite_dir(_)),
        assert(test_suite_dir('./')).

change_test_suite_dir :-
        repeat,
        test_suite_dir(CurDir), remslash(CurDir, DirName),
        write('The current test suite directory is:  '), write(DirName), nl,
        ask_user('New directory? (press Return to leave it as is) : ', NewDir),
        change_test_suite_dir(NewDir).

change_test_suite_dir('').
change_test_suite_dir(Dir) :-
        directory_exists(Dir),
        addslash(Dir, NewDir),
        retractall(test_suite_dir(_)),
        assert(test_suite_dir(NewDir)),
        write('Ok, the test suite directory is now:  '), write(Dir), nl.
change_test_suite_dir(Dir) :-
        write('Directory "'), write(Dir), write('" does not exist.  Let''s try again.'), nl, fail.

% append forward slash '/' = char(47) to the end of a string but only if it isn't there already
addslash(Dir, NewDir) :-
        atom(Dir), var(NewDir), name(Dir, DirL), last_byte(DirL, 47), NewDir = Dir, !.
addslash(Dir, NewDir) :-
        atom(Dir), var(NewDir), name(Dir, DirL), append(DirL, [47], NewDirL), name(NewDir, NewDirL), !.

% remove forward slash '/' = char(47) at the end of a string but only if it there
remslash(Dir, NewDir) :-
        atom(Dir), name(Dir, DirL), append(NewDirL, [47], DirL), name(NewDir, NewDirL), !.
remslash(Dir, NewDir) :-
        atom(Dir), var(NewDir), NewDir = Dir, !.

% return the last element of a list
last_byte([], _) :- fail, !.
last_byte([X], X).
last_byte([_|T], C) :- last_byte(T,C).

% relate test directory names to menu items
init_suite_list :-
```

```
        test_suite_dir(Dir),
        directory_members_of_directory(Dir, Set),
        retractall(suite(_,_)),
        assert_test_dirs(Set, 1).

assert_test_dirs([], _).
assert_test_dirs([Dir-_|DirList], N) :-
        assertz(suite(N, Dir)),
        NextN is N + 1,
        assert_test_dirs(DirList, NextN).

suite(S) :- suite(_, S).

/*
suite(quirk, 1, 'ch2').
suite(quirk, 2, 'ch2-r').
suite(quirk, 3, 'ch2-star').
suite(quirk, 4, 'ch3').
suite(quirk, 5, 'ch3-star').
suite(quirk, 6, 'ch4').
suite(quirk, 7, 'ch4-star').
suite(tsnlp, 8, 'tsnlp').
suite(tsnlp, 9, 'tsnlp-r').
suite(test, 10, 'test1').
suite(test, 11, 'test2').
suite(test, 12, 'badtest').
*/


%
%        F I L E   N A M E   C O N S T R U C T I O N
%

suite_files(Suite, Sent, Dict, DictAdd, OutStructs, ScriptFile, BibleTrees) :-
        get_suite_file_path(Suite, '-sent.txt', Sent),
        get_suite_file_path(Suite, '-dict.txt', Dict),
        dipett_default_file(Suite, dict_add, DictAdd),
        dipett_default_file(Suite, log, ScriptFile),
        get_out_file_name(Suite, OutStructs),
        get_bible_file_name(Suite, BibleTrees), !.

get_bible_file_name(Suite, BibleTreeFile) :-
        get_suite_file_path(Suite, '-test-trees.pl', BibleTreeFile), !.

get_out_file_name(Suite, OutFileName) :-
        dipett_default_file(Suite, parse_tree, OutFileName), !.

get_suite_file_path(Suite, FileSuffix, Path) :-
```

```
        construct_suite_filename(Suite, FileSuffix, FileName),
        construct_path(Suite, FileName, Path).

construct_suite_filename(SuiteName, FileSuffixName, FileName) :-
        name(SuiteName, Suite),                 % [ch2]
        name(FileSuffixName, FileSuffix),         % [-sent.txt]
        append(Suite, FileSuffix, File),        % ch2-sent.txt
        name(FileName, File).

construct_path(SuiteName, FileName, RelPath) :-
        test_suite_dir(PathName),                  %
        name(PathName, Path),                        % [tests/]
        name(SuiteName, Suite),                 % [ch2]
        name(FileName, File),                        % [ch2-dict.txt]
        append(Path, Suite, N1),                   % tests/ch2
        append(N1, [47], N2),                        % tests/ch2/
        append(N2, File, N3),                        % tests/ch2/ch2-dict.txt
        name(RelPath, N3).

dipett_default_file(Suite, dict_add, Path) :-
        check_add_filename_defaults(defaults, FileName),
        construct_path(Suite, FileName, Path).

dipett_default_file(Suite, parse_tree, Path) :-
        check_parse_tree_filename_defaults(defaults, FileName),
        construct_path(Suite, FileName, Path).

dipett_default_file(Suite, log, Path) :-
        check_log_filename_defaults(defaults, FileName),
        construct_path(Suite, FileName, Path).

%
%       C H E C K   I N P U T   F I L E S   E X I S T
%

input_files_exist(parse, Suite) :-
        suite_files(Suite, Sent, Dict, _Add, _Out, _Log, _Bible),
        check_exists(Suite, 'sentence', Sent), !,
        check_exists(Suite, 'aux_dict', Dict), !.

input_files_exist(compare_tree, Suite) :-
        suite_files(Suite, _Sent, _Dict, _Add, Out, _Log, _Bible),
        check_exists(Suite, 'parse_tree', Out), !.

input_files_exist(compare_bible, Suite) :-
        suite_files(Suite, _Sent, _Dict, _Add, Out, _Log, Bible),
        check_exists(Suite, 'parse_tree', Out), !,
        check_exists(Suite, 'bible_tree', Bible), !.
```

```
check_exists(_Suite, _FileType, FileName) :- file_exists(FileName).
check_exists(Suite, FileType, FileName) :-
        write('Sorry, '), write(Suite), write(' '), write(FileType),
        write(' file does not exist ('),
        write(FileName), write(')'), nl, fail.



/*****************************************************************************/
/* file: deval_bootdipett.pl                                                 */
/* version: April, 1999                                                      */
/* author: E. Scarlett                                                       */
/*****************************************************************************/

%
%  DIPETT must be loaded for much of this to work...
%

% this is from "go" and "dipett" in dipett/DIPETT_1_DRIVER.PL
% and also "proceed_with_all_defaults" and "proceed_with_batch_parsing" from conf/DIPETT_HAIKU_CUST.PL
boot_dipett :-
        % check if dipett is even here...
        check_dipett_loaded,

        % setup DIPETT and make it look like it's going normally with the default options
        get_dipett_version, get_haiku_version,
        retractall(reader_mode(_)),
        assert(reader_mode(off)),
        retractall(personal_defaults(_)),
        assert(personal_defaults(defaults)),
            msg([nl, 'Using standard default options !', nl]),
        affiche_version_et_copyright,
        menu_conven,

        % prepare to parse -- deal with the log file and set a time limit
         retract_previous_interface_assertions,
         assert(user:batch_parsing(on)),
        my_set_cpu_time_limit(120).


run_dipett_session(_Suite, InFile, AuxDict, DictAddFile, OutStructFile) :-
        % make sure the inputs are reasonable
        atom(InFile), atom(AuxDict), atom(DictAddFile), atom(OutStructFile),
        file_must_exist(InFile),
        file_must_exist(AuxDict),

        % set the standard batch parsing only defaults
```

```
        % except, we'll just overwrite existing files without prompting
        initialize_counters_to_zero,
        batch_parser_only_defaults(defaults),
        suppress_overwrite_prompt(Overwrite),

        % open output files and the sentences, and load the aux dict
        load_aux_dict(AuxDict),
            open_outstructs_file(OutStructFile),
        open_dict_add_file(DictAddFile),

        % parse the file
        deval_dipett_parse(InFile),
        my_process(7),

        % close the output files then reset dictionary state and overwrite defaults
        close_dict_add_file(DictAddFile),
        close_outstructs_file(OutStructFile),
        unload_aux_dict(AuxDict),
        restore_overwrite_prompt(Overwrite).

run_dipett_session(Suite, _, _, _, _) :-
        msg([deval, 'Oops, dipett session failed on suite  ',Suite, nl, nl]).

%
%       S U P P R E S S   O V E R W R I T E   P R O M P T
%
suppress_overwrite_prompt(Overwrite) :-
        check_overwrite_default(defaults, Overwrite),
        retractall(check_overwrite_default(defaults, _)),
        assert(check_overwrite_default(defaults, on)).
suppress_overwrite_prompt(_) :-
        retractall(check_overwrite_default(defaults, _)),
        assert(check_overwrite_default(defaults, on)).

restore_overwrite_prompt(Overwrite) :-
        atom(Overwrite),
        retractall(check_overwrite_default(defaults, _)),
        assert(check_overwrite_default(defaults, Overwrite)).
restore_overwrite_prompt(_) :-
        retractall(check_overwrite_default(defaults, _)),
        assert(check_overwrite_default(defaults, off)).

%
%       O P E N   O U T S T R U C T S   F I L E
%

% from dipett's want_to_savelb(_)
```

```
open_outstructs_file(FileName4) :-
        close_outstructs_file(FileName4),
        fail.
open_outstructs_file(FileName4) :-
        msg([deval, 'Opening output structures file - ', FileName4, nl]),
              check_parse_tree_filename(FileName4, FileName4_ok),
              new_file_ok(defaults, FileName4_ok),
              retractall(parse_tree_file(_)),
        retractall(parse_tree_filename(_)),
              open(FileName4_ok, write, Stream4),
              write_date_and_time(Stream4, FileName4_ok),
              assert(parse_tree_file(Stream4)),
              assert(parse_tree_filename(FileName4_ok)).
open_outstructs_file(FileName4) :-
        msg([deval, 'Failed to open output structures file - ', FileName4, nl]),
        fail, !.

close_outstructs_file(FileName4) :-
        clause(user:parse_tree_file(Stream4), true),
        msg([deval, 'Closing output structures file - ', FileName4, nl]),
              flush_output(Stream4),
        close(Stream4),
              retractall(parse_tree_file(_)),
        retractall(parse_tree_filename(_)).
close_outstructs_file(_).    % it doesn't exist, succeed.

%
%       O P E N   S C R I P T   F I L E
%

% from dipett's want_to_savel(_)

open_script_file(FileName1) :-
        close_script_file(FileName1),
        fail.
open_script_file(FileName1) :-
        msg([deval, 'Opening DIPETT log file - ', FileName1, nl]),
        suppress_overwrite_prompt(Overwrite),
              check_log_filename(FileName1, FileName1_ok),
              new_file_ok(defaults, FileName1_ok),
              write_to_save_file(FileName1_ok),
        clause(user:save_file(Stream1), true),
              write_date_and_time(Stream1, FileName1),
        restore_overwrite_prompt(Overwrite).
open_script_file(FileName1) :-
        msg([deval, 'Failed to open DIPETT log file - ', FileName1, nl]),
        fail, !.
```

```
close_script_file(FileName1) :-
        clause(user:save_file(Stream1), true),
        msg([deval, 'Closing DIPETT log file - ', FileName1, nl]),
              write_date_and_time(Stream1, FileName1),
              flush_output(Stream1),
        close(Stream1),
              retractall(save_file(_)),
        retractall(save_filename(_)).
close_script_file(_).    % it doesn't exist, succeed.


%
%       O P E N   D I C T I O N A R Y   A D D I T I O N S   F I L E
%

% from want_to_save2(_)
open_dict_add_file(FileName2) :-
        close_dict_add_file(FileName2),
              fail.
open_dict_add_file(FileName2) :-
        msg([deval, 'Opening dictionary additions file - ', FileName2, nl]),
        check_add_filename(FileName2, FileName2_ok),
            new_file_ok(defaults, FileName2_ok),
            write_to_dict_add_file(FileName2_ok).
open_dict_add_file(FileName2) :-
        msg([deval, 'Failed to open dictionary additions file - ', FileName2, nl]),
        fail, !.

close_dict_add_file(FileName2) :-
        clause(user:dict_add_filename(FN), true),
        msg([deval, 'Closing dictionary additions file - ', FileName2, nl]),
              open(FN, append, Stream2),
              nl(Stream2), write(Stream2, '% # of new words (entries): '),
              ctr_is(new_words, NW), write(Stream2, NW), nl(Stream2),
              flush_output(Stream2),
              retractall(dict_add_file(_)),
        retractall(dict_add_filename(_)).
close_dict_add_file(_FileName2).     % it doesn't exist, succeed.


%
%            L O A D   A U X   D I C T
%

% from want_to_save3(N)

load_aux_dict(DictName) :-
              file_must_exist(DictName), !,
```

```
        msg([deval, 'Loading auxilliary dictionary - ', DictName, nl]),
        consult(DictName).

%
%           S E T U P   F O R   P A R S I N G   A N D   G O   P A R S E
%

% from set_cpu_time_limit
my_set_cpu_time_limit(LIMIT) :-
        number(LIMIT),
        LIMIT > 0,
             LIMITmsec is LIMIT * 1000,
        assert(cpu_time_limit(LIMITmsec)).
my_set_cpu_time_limit(_) :-
             LIMITmsec is 300 * 1000,
        assert(cpu_time_limit(LIMITmsec)).


% from goto_desired_unit
setup_to_parse_all_units(FileName) :-
        msg([deval, 'Opening test sentences - ', FileName, nl]),
        check_unit_number(1, '', Num_low),
            check_unit_number(1, '', NumT_ok),
        check_unit_number(2, '', Num_up),
            Num_up >= Num_low, !,
            NumT_minus_1 is NumT_ok - 1,  % will be incremented!
            ctr_set(units_nb, NumT_minus_1),
            retractall(upper_string_number(_)),
        assert(upper_string_number(Num_up)),
            see(FileName),
        goto_unit(Num_low).
setup_to_parse_all_units(FileName) :-
        msg([deval, 'Failed to open file - ', FileName, nl]),
        fail, !.

% from process(2a)
deval_dipett_parse(FileName) :-
        file_must_exist(FileName),
        setup_to_parse_all_units(FileName),
        msg([deval, '[Files opened successfully -- starting DIPETT parse_all]', nl]), nlp,
        parse_all(FileName, straight).
deval_dipett_parse(_) :-
        msg([deval, '[Failed to parse sentences]', nl]).

%
%       W R I T E   C P U   T I M E   L I M I T
%
% writes the current cpu time limit
```

```
write_cpu_time_limit :-
        cpu_time_limit(LIMITmsec),
             LIMITsec is LIMITmsec / 1000,
           msg([deval, 'CPU time limit is ', LIMITsec, ' seconds', nl]).
write_cpu_time_limit :- !.

%
%       W R I T E   D A T E   A N D   T I M E  /  0
%
% writes the date and time e.g. [date&time: 23/4/99 14:30:49]
write_date_and_time :-
   date(date(Y,M0,D)), M is M0 + 1, time(time(H,Mi,Se)),
   msg(['  [', D, '/', M, '/', Y, ' ', H, ':']),
   ((Mi < 10, msg(['0'])) ; Mi >= 10), msg([Mi, ':']),
   ((Se < 10, msg(['0'])) ; Se >= 10), msg([Se, ']']).

%
%       M Y   P R O C E S S   7
%

/* ..................................................................... */

/*  #7: End of session; produce statistics.                              */
/*  this version is exactly the same as DIPETT_1_DRIVER.PL except that it    */
/*  doesn't call byebye on process(7)                                    */

my_process(X) :- X == 7,
             nlp(X), nlp(X),
             ((X == 7, writep(X,'END OF PARSING SESSION STATISTICS:  ')) ;
              (X \== 7, writep(X,'ONGOING PARSING SESSION STATISTICS:  '))),
             date(date(Y,M0,D)), M is M0 + 1, time(time(H,Mi,S)),
             writep(X,'[date&time: '),
             writep(X,D), writep(X,'/'), writep(X,M), writep(X,'/'),
             writep(X,Y), writep(X,' '), writep(X,H), writep(X,':'),
             ((Mi < 10, writep(X,'0')) ; Mi >= 10),
             writep(X,Mi), writep(X,':'),
             ((S < 10, writep(X,'0')) ; S >= 10), writep(X,S), writep(X,']'),
             nlp(X),

             ((clause(user:cpu_time_limit(TL),true),
               writep(X,'   0- cpu time limit: '),
               superv_cpu_time_2(X,TL)) ;
              \+ clause(user:cpu_time_limit(_),_)),

             ctr_is(units_total,Nb_Units),
             writep(X,'   1- Total nb. of strings processed: '),
             writep(X,Nb_Units), nlp(X),
             ctr_is(total_nb_tokens,Total_Nb_Tokens),
```

```
writep(X,'      (with an average nb. of tokens of: '),
((Nb_Units > 0,
  AV_Nb_Tokens is Total_Nb_Tokens / Nb_Units) ;
 (Nb_Units == 0,
  AV_Nb_Tokens = 0)),
writep(X,AV_Nb_Tokens), writep(X,')'),
nlp(X),

ctr_is(nb_fragments_parsed,Nb_Fragments),
writep(X,'   2- Total nb. of strings parsed as fragments: '),
writep(X,Nb_Fragments), writep(X,' ('),
percent(Nb_Fragments,Nb_Units,P1),
writep(X,P1), writep('%)'), nlp(X),
ctr_is(total_nb_fragments,Total_Nb_Fragments),
writep(X,'      (with an average nb. of fragments of: '),
((Nb_Fragments > 0,
  AV_Nb_Frags is Total_Nb_Fragments / Nb_Fragments) ;
 (Nb_Fragments == 0,
  AV_Nb_Frags = 0)),
writep(X,AV_Nb_Frags), writep(X,')'), nlp(X),
writep(X,'      (partial parsing mode [quick/standard]: '),
((quick_frag_parsing(on), writep(X,'quick)')) ;
 (quick_frag_parsing(off), writep(X,'standard)') ;
 writep(X,'N/A'))),
nlp(X),

ctr_is(bad_units,Nb_Bad_Units),
writep(X,'   3- Total nb. of strings not parsed: '),
writep(X,Nb_Bad_Units), writep(X,' ('),
percent(Nb_Bad_Units,Nb_Units,P2),
writep(X,P2), writep('%)'), nlp(X),

ctr_is(skip_units,Nb_Skip_Units),
writep(X,'   4- Total nb. of strings skipped: '),
writep(X,Nb_Skip_Units), writep(X,' ('),
percent(Nb_Skip_Units,Nb_Units,P3),
writep(X,P3), writep('%)'), nlp(X),

writep(X,'   5- Total nb. of strings fully parsed: '),
Nb_Ideal_Units is
  Nb_Units - (Nb_Fragments + Nb_Bad_Units + Nb_Skip_Units),
writep(X,Nb_Ideal_Units), writep(X,' ('),
percent(Nb_Ideal_Units,Nb_Units,P4),
writep(X,P4), writep('%)'), nlp(X),

Nb_Units_Parsed is (Nb_Units - Nb_Skip_Units) - Nb_Bad_Units,
Nb_Units_ok is Nb_Units - Nb_Skip_Units,
writep(X,'   6- Percent of successful parses (full or frag.,'),
```

```
writep(X,' excluding #4): '),
percent(Nb_Units_Parsed,Nb_Units_ok,P5),
writep(X,P5), writep('%'), nlp(X),

ctr_is(total_new_words,Nb_New_Words),
writep(X,'   7- Total nb. of dictionary additions: '),
writep(X,Nb_New_Words), nlp(X),

ctr_is(total_cpu_time,TCT),
writep(X,'   8- Total cpu time (parsing & other tasks): '),
superv_cpu_time_2(X,TCT),

writep(X,'   9- Average cpu time per string processed: '),
((TCT > 0, AV_CPU is TCT / Nb_Units) ; (TCT == 0, AV_CPU = 0)),
superv_cpu_time_2(X,AV_CPU),

(quintus ->
 writep(X,'   *- Prolog version: Quintus.') ;
 writep(X,'   *- Prolog version: SICStus.')),
nlp(X), nlp(X),


((X \== '7a',
  ((clause(user:save_filename(FileName1),true),
    writep(X,'Upon request, output has been copied in file: '),
    writep(X,FileName1), nlp(X)) ;
   \+ clause(user:save_filename(_),_)),

  ((clause(user:parse_tree_filename(FileName4),true),
    writep(X,'Upon request, output structures have been'),
    writep(X,' copied in file: '), writep(X,FileName4),
    nlp(X)) ;
   \+ clause(user:parse_tree_filename(_),_)),

  ((clause(user:tagger_filename(FileName9),true),
    writep(X,'Upon request, tags have been copied in'),
    writep(X,' file: '), writep(X,FileName9), nlp(X)) ;
   \+ clause(user:tagger_filename(_),_)),

  ((clause(user:dict_add_filename(FileName2),true),
    writep(X,'Dictionary additions (if any) '),
    writep(X,'have been copied in file: '),
    writep(X,FileName2)) ;
   \+ clause(user:dict_add_filename(FileName2),_)),

  nlp(X)) ;

 X == '7a'),
```

```
        ((X == 7, nlp(X)) ;
          X == '7a' ;
         (X == '7b',
          nlp(X),
          writep(X,'PARSING SESSION ... continued elsewhere!'),
          nlp(X), nlp(X))),

        !.


/*************************************************************************/
/* file: devalrw.pl                                                      */
/* version: April, 1999                                                  */
/* copied directly from DIPETT_1_DRIVER.PL                               */
/*************************************************************************/


% This file is loaded _only_ if dipett is not already present

:- dynamic check_overwrite_default/2.
:- retractall(reader_mode(_)), assert(reader_mode(off)).

/*************************************************************************/
/* the 'READ' stuff ...                                                  */

read_answer(Answer) :-
   reader_mode(on), !,
   ttyflush,
   see('../Bin/DIPETT_ANSWER'),
   read_line(Line), name(Answer,Line), writep2(Answer), nlp2,
   seen,
   nl, nl, write('The answer is: ('), write(Answer), write(')'), nl, nl,
   ttyflush.

read_answer(Answer) :-
   reader_mode(off),
   see(user),
   read_line(Line), name(Answer,Line), writep2(Answer), nlp2.


read_line(Line)             :- get0(Char), read_line(Char,Line).
read_line(10,[])            :- !.  % 10: line feed
read_line(Char,[Char|Line]) :- get0(Char1), read_line(Char1,Line).

/*************************************************************************/
/* the 'WRITE' stuff ...                                                 */
```

```
/*                                                                    */
/* Note: Stream1 is used for keeping a log of the parsing session, and */
/*       Stream2 is used for recording dictionary additions.          */

/* NLP always affects the default output stream (i.e. terminal) and   */
/* possibly Stream1 if it is opened (for keeping a log of the session). */
nlp :- clause(user:save_file(Stream1),true), nl(Stream1), nl(user), !.
nlp :- nl(user).


/* NLP2 only affects Stream1 (if it is opened).                       */
nlp2 :- clause(user:save_file(Stream1),true), nl(Stream1), !.
nlp2.


/* WRITEP is akin to NLP.                                             */
writep(X) :- clause(user:save_file(Stream1),true), write(Stream1,X),
             write(user,X), !.
writep(X) :- write(user,X).

writep_1f(X) :- clause(user:save_file(Stream1),true), format(Stream1,'~1f',X),
                format(user,'~1f',X), !.
writep_1f(X) :- format(user,'~1f',X).



/* WRITEP2 is akin to NLP2.                                           */
writep2(X) :- clause(user:save_file(Stream1),true), write(Stream1,X), !.
writep2(_).

writep2_1f(X) :-
   clause(user:save_file(Stream1),true), format(Stream1,'~1f',X), !.
writep2_1f(_).



/* TABP is akin to NLP2.                                              */
tabp(X) :-
   clause(user:save_file(Stream1),true), tab(Stream1,X), tab(user,X), !.
tabp(X) :-
   tab(user,X).


/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

new_file_ok(Defaults,FileName) :-
   nonvar(Defaults),
   file_exists(FileName,write),
```

```
   check_overwrite_default(Defaults,on),  % Specified in one's config. file.
   delete_file(FileName), !.
new_file_ok(_,FileName)        :-
   file_exists(FileName,write),
   writep('        !! This file already exists ... OVERWRITE IT [y/N]? '),
   read_answer(OVER),
   ((OVER == y,
     delete_file(FileName), !) ;
    (OVER \== y,
     writep('           then, please enter a new filename ...'), nlp,
     !, fail)).
new_file_ok(_,FileName)        :-
   \+ file_exists(FileName,write).

/*****************************************************************************/
/* WRITE_DATE_AND_TIME */

write_date_and_time(S,Filename) :-
   write(S,'% '), write(S,Filename), write(S,' '),
   date(date(Y,M0,D)), M is M0 + 1, time(time(H,Mi,Se)),
   write(S,'[date&time: '), write(S,D), write(S,'/'), write(S,M),
write(S,'/'),
   write(S,Y), write(S,' '), write(S,H), write(S,':'),
   ((Mi < 10, write(S,'0')) ; Mi >= 10), write(S,Mi), write(S,':'),
   ((Se < 10, write(S,'0')) ; Se >= 10), write(S,Se), write(S,']'),
   nl(S), nl(S), flush_output(S).


/*****************************************************************************/
/********************** DEFAULT NAMES FOR OUTPUT FILES ***********************/
/**********************    from DIPETT_HAIKU_CUST.PL   ***********************/

% tagger output file
check_tag_filename_defaults(defaults, 'zzz.DIPETT-tagger.pl').

% session log file
check_log_filename_defaults(defaults, 'zzz.DIPETT-HAIKU.script').

% output structures file (parse trees & case structures)
check_parse_tree_filename_defaults(defaults, 'zzz.DIPETT-HAIKU.OutStructs.pl').

% dictionnary additions file
check_add_filename_defaults(defaults, 'zzz.DIPETT-dict_add.pl').

% Do ask whether an existing file should be overwritten!
check_overwrite_default(defaults, off).

/*****************************************************************************/
```

Appendix E: Comparison of DIPETT's Performance on the TSNLP Phenomena

Appendix E shows the changes in DIPETT's performance on the detailed TSNLP phenomena. The sentences counted are those that were affected by the grammar changes — sentences that produced fragmentary parses with DIPETT v3.0 but parsed successfully with DIPETT v4.0, and sentences that parsed successfully with both parsers but the parse trees were not identical.

Phenomenon	Sentence Count	DIPETT v3.0			DIPETT v4.0		Increase in Number of	
		Fragmentary Parses	Correct Full Parses	Incorrect Full Parses	Correct Full Parses	Incorrect Full Parses	Full parse trees	Correct Parse Trees
C_Agreement-Coordinated_subj(correlatives)	6	6	0	0	6	0	6	6
C_Complementation-Divalent-Complementizer(that)_subjunctive/should_bare_infinitive	1	1	0	0	1	0	1	1
C_Complementation-Tetravalent	1	1	0	0	1	0	1	1
C_Complementation-Trivalent-Direct_object_as_VP_ing	1	1	0	0	1	0	1	1
C_Diathesis-Active-Divalent-Complementizer(that)_subjunctive/should_bare_infinitive	1	1	0	0	1	0	1	1
C_Diathesis-Active-Tetravalent	1	1	0	0	1	0	1	1
C_Diathesis-Active-Trivalent-Direct_object_as_VP_ing	1	1	0	0	1	0	1	1
C_Diathesis-Passive-Auxiliary(be)	1	0	0	1	1	0	0	1
C_Diathesis-Passive-Auxiliary(get)	2	2	0	0	2	0	2	2
C_Diathesis-Passive-Divalent-As_VP_ing	2	2	0	0	2	0	2	2
C_Diathesis-Passive-Divalent-Prep	1	1	0	0	1	0	1	1
C_Diathesis-Passive-Divalent-VP_ed	4	2	1	1	4	0	2	3
C_Diathesis-Passive-Monovalent-Subj(NP)	2	0	0	2	2	0	0	2
C_Diathesis-Passive-No_agent	4	2	0	2	4	0	2	4
C_Diathesis-Passive-Obl_agent	1	0	0	1	1	0	0	1
C_Diathesis-Passive-Tetravalent-VP_ed_particle_PP	1	1	0	0	1	0	1	1
C_Diathesis-Passive-Trivalent-VP_ed_compl(that)_subjunctive/should/bare_infinitive	1	1	0	0	1	0	1	1
C_Diathesis-Passive-Trivalent-VP_ed_compl(that)_subjunctive/should_bare_infinitive	1	1	0	0	1	0	1	1
C_Diathesis-Passive-Trivalent-VP_ed_particle	2	0	0	2	2	0	0	2
C_Negation-Tense_aspect_modality-BE_HAVE-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-BE_HAVE-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-CAN-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-CAN-Present_tense	14	8	0	6	14	0	8	14
C_Negation-Tense_aspect_modality-Contractions-BE_HAVE-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-Contractions-BE_HAVE-Present_tense	6	5	0	1	6	0	5	6
C_Negation-Tense_aspect_modality-Contractions-CAN-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-Contractions-CAN-Present_tense	7	4	0	3	7	0	4	7
C_Negation-Tense_aspect_modality-Contractions-DARE-Present_tense	8	8	0	0	8	0	8	8
C_Negation-Tense_aspect_modality-Contractions-MAY-Past_tense	4	2	0	2	4	0	2	4

Phenomenon	Sentence Count	DIPETT v3.0			DIPETT v4.0		Increase in Number of	
		Fragmentary Parses	Correct Full Parses	Incorrect Full Parses	Correct Full Parses	Incorrect Full Parses	Full parse trees	Correct Parse Trees
C_Negation-Tense_aspect_modality-Contractions-MAY-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-Contractions-MUST-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-Contractions-NEED-Present_tense	7	6	0	1	7	0	6	7
C_Negation-Tense_aspect_modality-Contractions-OUGHT-Present_tense	6	4	0	2	6	0	4	6
C_Negation-Tense_aspect_modality-Contractions-SHALL-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-Contractions-SHALL-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-Contractions-USED-Past_tense	24	16	0	8	24	0	16	24
C_Negation-Tense_aspect_modality-Contractions-WILL-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-Contractions-WILL-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-DARE-Past_tense	8	8	0	0	8	0	8	8
C_Negation-Tense_aspect_modality-DARE-Present_tense	7	6	0	1	7	0	6	7
C_Negation-Tense_aspect_modality-MAY-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-MAY-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-MUST-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-NEED-Present_tense	7	6	0	1	7	0	6	7
C_Negation-Tense_aspect_modality-OUGHT-Present_tense	6	4	0	2	6	0	4	6
C_Negation-Tense_aspect_modality-SHALL-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-SHALL-Present_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-USED-Past_tense	23	12	2	9	23	0	12	21
C_Negation-Tense_aspect_modality-WILL-Past_tense	4	2	0	2	4	0	2	4
C_Negation-Tense_aspect_modality-WILL-Present_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-BE_HAVE-Past_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-BE_HAVE-Present_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-CAN-Past_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-CAN-Present_tense	7	4	0	3	7	0	4	7
C_Tense-Aspect-Modality-Contractions-BE_HAVE-Past_tense	2	1	0	1	2	0	1	2
C_Tense-Aspect-Modality-Contractions-BE_HAVE-Present_tense	3	2	0	1	3	0	2	3
C_Tense-Aspect-Modality-Contractions-CAN-Past_tense	2	1	0	1	2	0	1	2
C_Tense-Aspect-Modality-Contractions-MAY-Past_tense	2	1	0	1	2	0	1	2
C_Tense-Aspect-Modality-Contractions-MAY-Present_tense	2	1	0	1	2	0	1	2
C_Tense-Aspect-Modality-Contractions-MUST-Present_tense	2	1	0	1	2	0	1	2
C_Tense-Aspect-Modality-Contractions-SHALL-Past_tense	2	1	0	1	2	0	1	2
C_Tense-Aspect-Modality-Contractions-WILL-Past_tense	6	3	0	3	6	0	3	6
C_Tense-Aspect-Modality-Contractions-WILL-Present_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-MAY-Past_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-MAY-Present_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-MUST-Present_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-OUGHT-Present_tense	6	4	0	2	6	0	4	6
C_Tense-Aspect-Modality-SHALL-Past_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-SHALL-Present_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-USED-Past_tense	8	4	1	3	8	0	4	7

Phenomenon	Sentence Count	DIPETT v3.0			DIPETT v4.0		Increase in Number of	
		Fragmentary Parses	Correct Full Parses	Incorrect Full Parses	Correct Full Parses	Incorrect Full Parses	Full parse trees	Correct Parse Trees
C_Tense-Aspect-Modality-WILL-Past_tense	4	2	0	2	4	0	2	4
C_Tense-Aspect-Modality-WILL-Present_tense	4	2	0	2	4	0	2	4
NP_Modification-Relative_clauses-Non_restrictive-Genitive-Collectives	3	2	0	1	1	2	2	1
NP_Modification-Relative_clauses-Non_restrictive-Genitive-Nonpersonal	3	2	0	1	1	2	2	1
NP_Modification-Relative_clauses-Non_restrictive-Genitive-Personal	3	2	0	1	1	2	2	1
NP_Modification-Relative_clauses-Non_restrictive-Object_extraction-Collectives	2	1	0	1	2	0	1	2
NP_Modification-Relative_clauses-Non_restrictive-Object_extraction-Nonpersonal	2	1	0	1	2	0	1	2
NP_Modification-Relative_clauses-Non_restrictive-Object_extraction-Personal	3	2	0	1	3	0	2	3
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction(stranded)-Collectives	2	1	0	1	2	0	1	2
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction(stranded)-Nonpersonal	2	1	0	1	2	0	1	2
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction(stranded)-Personal	3	2	0	1	3	0	2	3
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction-Collectives	29	27	1	1	29	0	27	28
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction-Nonpersonal	27	26	0	1	27	0	26	27
NP_Modification-Relative_clauses-Non_restrictive-PP_extraction-Personal	27	26	0	1	27	0	26	27
NP_Modification-Relative_clauses-Non_restrictive-Subject_extraction-Collectives	2	2	0	0	2	0	2	2
NP_Modification-Relative_clauses-Non_restrictive-Subject_extraction-Nonpersonal	2	2	0	0	2	0	2	2
NP_Modification-Relative_clauses-Non_restrictive-Subject_extraction-Personal	2	2	0	0	2	0	2	2
S_Types-Questions-Wh-Mod(DET_wh+ADV)	1	1	0	0	0	1	1	0
S_Types-Questions-Wh-Pred(DET_wh+NP)	1	1	0	0	0	1	1	0
S_Types-Questions-Wh-Subj(DET_wh+NP)	3	3	0	0	0	3	3	0
S_Types-Questions-Y/N_questions-Inverted	2	0	2	0	2	0	0	0
S_Types-Questions-Y/N_questions-Negative-Inverted-Non-tagged	3	1	2	0	2	1	1	0
S_Types-Questions-Y/N_questions-Non_inverted-Non-tagged	13	13	0	0	13	0	13	13
Total	456	312	11	133	443	13	312	433

Appendix F: Re-evaluation of DIPETT's Performance on the Quirk Sentences

Appendix F shows the performance of DIPETT v4.0 on the test suite that was extracted from Quirk, compared to the performance of DIPETT v3.0 on the same sentences. The sentences shown are those that were affected by the grammar changes — sentences that produced fragmentary parses with DIPETT v3.0 but parsed successfully with DIPETT v4.0, and sentences that parsed successfully with both parsers but the parse trees have changed.

✓ Indicates that the parse tree representation is acceptable.
✗ Indicates that the parse tree representation is not acceptable.
? Indicates that it is questionable whether the parse tree is an improvement.

S Subject
V Verb
O Object
C Complement
A Adverb
{A} Subordinate adverbial, initial or final unless specifically marked medial

Quirk Sentences Parsed Successfully by DIPETT v4.0 but not DIPETT v3.0

Number	*Sentence*	DIPETT v3.0 *Parse*	*Grammar Change*	DIPETT v4.0 *Parse*
115. ✓	The ship may have been being sunk.	Fragmentary parse	"be being"	$V_{passive}$ may have been being sunk O the ship
165. ✓	He is being a nuisance again.	Fragmentary parse	"be being"	S he $V_{copular}$ is being C/A a nuisance {A} again
166. ✓	He is being naughty again.	Fragmentary parse	"be being"	S he $V_{copular}$ is being C/A *conj-adjp-implicit-and* naughty again
189. ?	Which cup is yours?	Fragmentary parse	*parse_tree__inter_decla_or_imper*	*parse_tree__inter_decla_or_imper det* which *head-noun* cup $V_{copular}$ is C/A yours
218. ✗	Have you a box of matches?	Fragmentary parse	*parse_tree__inter_decla_or_imper*	*parse_tree__inter_decla_or_imper* $V_{imperative}$ have O a box of matches O/C/A *iobj* you
220. ✗	Who has borrowed my pencil?	Fragmentary parse	*parse_tree__inter_decla_or_imper*	*parse_tree__inter_decla_or_imper* S *nom-cl* [*wh-interro-cl-who* V has] V borrowed O my pencil
257. ✗	Also, why doesn't he take the children abroad?	Timed out after 5 minutes	*parse_tree__inter_decla_or_imper*	*parse_tree__inter_decla_or_imper* {A} also S *nom-cl* [*wh-interro-cl-why* V doesn't O he] V take O the children {A} abroad
273. ✓	There was someone knocking at the door.	Fragmentary parse	S V O C/A	S there $V_{copular}$ was O someone C/A *adj* knocking *PP* at the door
284. ✓	They demanded that she call and see them	Fragmentary parse	subjunctive	S they V demanded O *nom-cl* [*full-that-cl* S she V *conj-verbs-and* call see O them]
302. ✓	He might have been being questioned by the	Fragmentary parse	"be being"	$S_{passive}$ the police $V_{passive}$ might have been being questioned O he

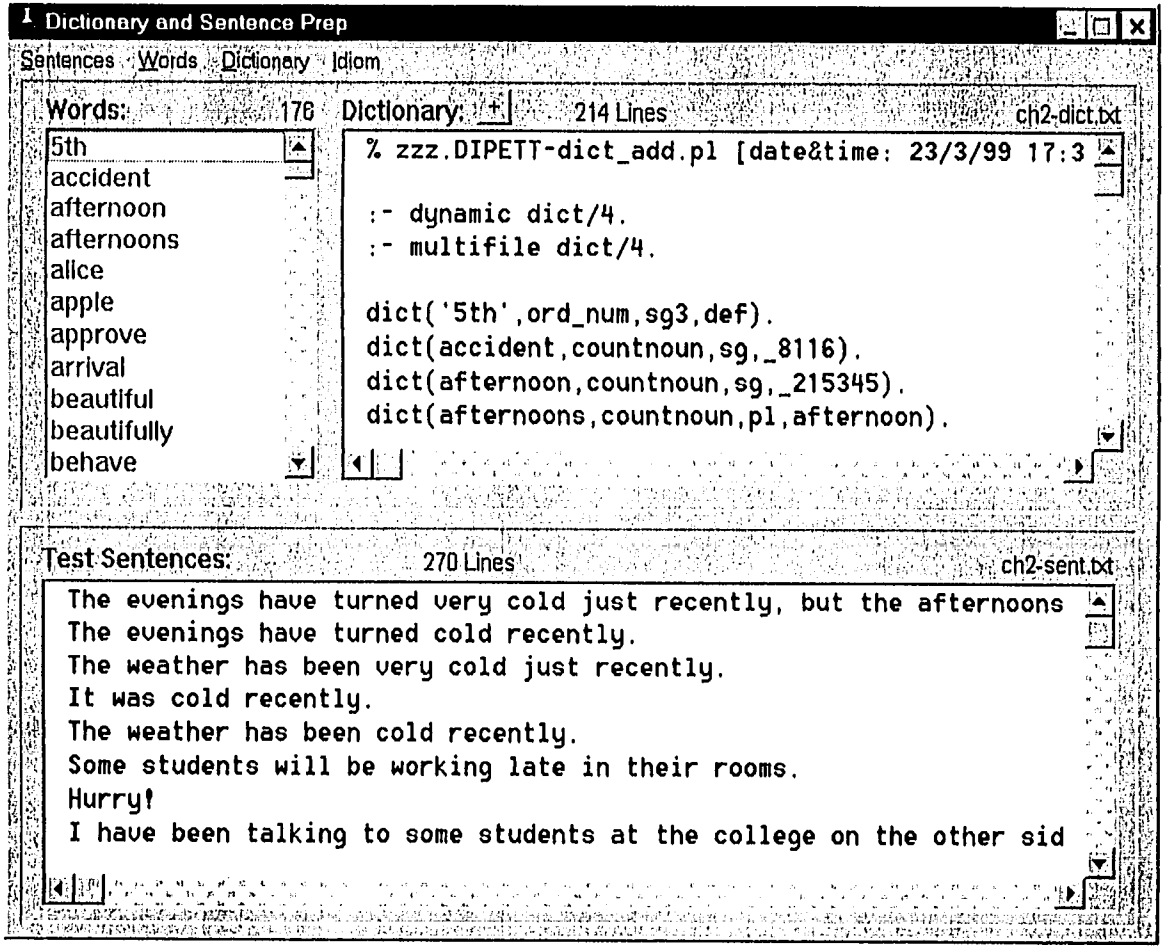
Number	*Sentence*	DIPETT v3.0 *Parse*	*Grammar Change*	DIPETT v4.0 *Parse*
	police.			
313. ✘	At no time was the entrance left unguarded.	Fragmentary parse	S V O C/A	{A} at no S time $V_{copular}$ was O the entrance C/A *conj-adjp-implicit-and* left ungarded
341. ✘	My parents are both working.	Fragmentary parse	S V O C/A	S my parents $V_{copular}$ are O both C/A *adj* working
388. ✘	Have you a lighter?	Fragmentary parse	*parse_tree__inter_decla_or_imper*	*parse_tree__inter_decla_or_imper* $V_{imperative}$ have O a lighter O/C/A *iobj* you
393. ✘	Have you any brothers?	Fragmentary parse	*parse_tree__inter_decla_or_imper*	*parse_tree__inter_decla_or_imper* $V_{imperative}$ have O any brothers O/C/A *iobj* you
447. ✔	He usen't to smoke.	Fragmentary parse	"use to"	S he V use to smoke *adv* not
519. ✔	Our team got beaten by the visitors.	Fragmentary parse	passive "get"	$S_{passive}$ the visitors $V_{passive}$ got beaten O our team
525. ✔	The cigars smoked here tend to be expensive.	Fragmentary parse		S the cigars *postmod [rel-cl* V smoked O the cigars] V tend *adv* here O *nom-cl [to-inf* $V_{copular}$ be C/A expensive]
529. ✔	It is important that he work hard.	Fragmentary parse	passive "get"	S it $V_{copular}$ is C/A important *nom-cl [full-that-cl* S he $V_{infinitive}$ work] {A} hard
535. ✔	He is being examined.	Fragmentary parse	"be being"	$V_{passive}$ is being examined O he
538. ✔	He may be being examined.	Fragmentary parse	"be being"	$V_{passive}$ may be being examined O he
539. ✔	He has been being examined.	Fragmentary parse	"be being"	$V_{passive}$ has been being examined O he
540. ✔	He may have been being examined.	Fragmentary parse	"be being"	$V_{passive}$ may have been being examined O he
548. ✔	I expected to be being interviewed then.	Fragmentary parse	"be being"	S I V expected O *nom-cl [ing-cl* $V_{copular}$ be being interviewed] {A} then
554. ✔	I saw her being interviewed.	Fragmentary parse	progressive "being"	S I V saw O *nom-cl [genitive-ing-cl deter-pron* her $V_{copular}$ being C/A interviewed]
562. ✔	The committee proposes that Mr_Day be elected.	Fragmentary parse	subjunctive	S the committee V proposes O *nom-cl [full-that-cl* S mr-day $V_{infinitive}$ be elected]
563. ✔	The committee proposed that Mr_Day be elected	Fragmentary parse	subjunctive	S the committee V proposed O *nom-cl [full-that-cl* S mr-day $V_{infinitive}$ be elected]
564. ✔	I demand that the committee reconsider its decision.	Fragmentary parse	subjunctive	S I V demand O *nom-cl [full-that-cl* S the committee $V_{infinitive}$ reconsider O its decision]
565. ✔	His sole requirement was that the system work.	Fragmentary parse	subjunctive	S his sole requirement $V_{copular}$ was C/A *nom-cl [full-that-cl* S the system $V_{infinitive}$ work]
566. ✔	They recommend that this tax be abolished.	Fragmentary parse	subjunctive	S they $V_{copular}$ recommend C/A *nom-cl [full-that-cl* $V_{infinitive}$ be abolished O this tax]
567. ✔	It is appropriate that this tax be abolished.	Fragmentary parse	subjunctive	S it $V_{copular}$ is C/A appropriate *nom-cl [full-that-cl* $V_{infinitive}$ be abolished O this tax]
592. ✔	She is being missed.	Fragmentary parse	"be being"	$V_{passive}$ is being missed O she
594. ✔	She may be being missed.	Fragmentary parse	"be being"	$V_{passive}$ may be being missed O she
595. ✔	She has been being missed.	Fragmentary parse	"be being"	$V_{passive}$ has been being missed O she
596. ✔	She may have been being missed.	Fragmentary parse	"be being"	$V_{passive}$ may have been being missed O she
597. ✔	The cat got run over.	Fragmentary parse	passive "get"	S the cat $V_{copular}$ got run {A} over
599. ✔	James got caught by the police.	Fragmentary parse	passive "get"	$S_{passive}$ the police $V_{passive}$ got caught O James
602. ✘	I don't want to get mixed up with the police again.	Fragmentary parse	passive "get"	S I V do want *adv* not O *nom-cl [to-inf* V get C/A mixed *PP [P* up with] the police] {A} again
634. ✔	My uncle got tired.	Fragmentary parse	passive "get"	$V_{passive}$ got tired O my uncle

Quirk Sentences Parsed Successfully by both DIPETT v3.0 and DIPETT v4.0 — parse trees are different

Number	*Sentence*	DIPETT v3.0 *Parse*	DIPETT v4.0 *Parse*
21. ✔	The weather has been very cold this month.	S the weather V_{active} has O *implicit-conj-adj* been very cold this *head-noun* month	S the weather $V_{copular}$ has been C/A *implicit-conj-adj* very cold this *head-noun* month
76. ✔	The accident was seen.	S the accident $V_{copular}$ was C/A seen	$V_{passive}$ was seen O the accident
89. ✔	The ladder must have been placed there.	S the ladder $V_{copular}$ must have been C/A *conj-adjs-implicit-and* placed there	$V_{passive}$ must have been placed O the ladder {A} there
90. ✔	The ladder must have been placed there by an intruder.	S the ladder $V_{copular}$ must have been C *conj-adjs-implicit-and* placed there *PP* by an intruder	$S_{passive}$ an intruder $V_{passive}$ must have placed O the ladder {A} there
105. ✔	Our team has been beaten.	S our team $V_{copular}$ has been C beaten	$V_{passive}$ has been beaten O our team
113. ✔	The ship has been sunk.	S the ship $V_{copular}$ has been C/A sunk	$V_{passive}$ has been sunk O the ship
199. ✔	The gold has been hidden somewhere.	S the gold $V_{copular}$ has been C/A *conj-adjs-implicit-and* hidden somewhere	$V_{passive}$ has been hidden O the gold {A} somewhere
262. ?	Don't be frightened!	$V_{imperative}$ do be *adv* not C/A frightened	$V_{imperative}$ do be frightened *adv* not
297. ✔	Will he be asked any questions?	*yes-no-question ques-aux-or-modal* will S he $V_{copular}$ be C/A *conj-adjs-implicit-and* asked any *head-noun* questions	*yes-no-question ques-aux-or-modal* will V be asked O any questions O he
352. ✔	There used to be a school on the island.	S there V used O *nom-cl* [*to-inf* $V_{copular}$ be C/A a school *postmod* [*PP* on the island]	$S_{stative}$ there V used to be C/A a school *postmod* [*PP* on the island]
355. ✘	The president will be met by thousands of people.	S the president $V_{copular}$ will be C/A *adj* met *PP* by thousands *postmod* [*PP* of people]	$S_{passive}$ thousands $V_{passive}$ will meet O *PP* of people O/C/A the president
358. ✔	You will be asked questions.	S you $V_{copular}$ will be C/A *adj* asked *head-noun* questions	$V_{passive}$ will be asked O questions O you
361. ✔	He used to read for hours.	S he V used to read O *PP* for hours	S he V used to read O *PP* for hours
377. ✔	Our team has never been beaten.	S our team $V_{copular}$ has been *adj* never C/A beaten	$V_{passive}$ has been beaten *adj* never O our team
381. ✔	They may have been eaten.	S they $V_{copular}$ may have been C/A *adj* eaten	$V_{passive}$ has been eaten O they
415. ✔	They can all come.	S they V can come *adv* all	S they V can come *adv* all
418. ✔	It can be done by Ann.	S it $V_{copular}$ can be C/A *adj* done *PP* by Ann	$S_{passive}$ Ann $V_{passive}$ can be done O it
445. ✔	She used to attend regularly.	S she $V_{present}$ used to attend {A} regularly	S she V_{past} used to attend {A} regularly
446. ✔	I used to be interested in birdwatching.	S i V used O *nom-cl* [*to-inf* $V_{copular}$ be C/A *adj* interested *PP* in birdwatching]	$V_{passive}$ used to be interested O *iobj PP* in birdwatching O i
448. ✔	He used not to smoke.	S he $V_{present}$ used to smoke *adv* not	S he V_{past} used to smoke *adv* not
449. ✔	He didn't use to smoke.	S he V did use *adv* not O *nom-cl* [*to-inf* V smoke]	S he V_{past} did use to smoke *adv* not
450. ✔	He didn't used to smoke.	S he V did {A} *subordinator* not [*past-part-phrase* $V_{passive}$ used OO *iobj PP* to smoke]	S he V_{past} did used to smoke *adv* not
532. ✔	He may be examined.	S he $V_{copular}$ may be C/A *adj* examined	$V_{passive}$ may be examined O he
534. ✔	He has been examined.	S he $V_{copular}$ has been C/A *adj* examined	$V_{passive}$ has been examined O he
537. ✔	He may have been examined.	S he $V_{copular}$ may have been C/A *adj* examined	$V_{passive}$ may have been examined O he
588. ✔	She is missed.	S she $V_{copular}$ is C/A *adj* missed	$V_{passive}$ is missed O she
589. ✔	She was missed.	S she $V_{copular}$ was C/A *adj* missed	$V_{passive}$ was missed O she
590. ✔	She may be missed.	S she $V_{copular}$ may be C/A *adj* missed	$V_{passive}$ may be missed O she
591. ✔	She has been missed.	S she $V_{copular}$ has been C/A *adj* missed	$V_{passive}$ has been missed O she
593. ✔	She may have been missed.	S she $V_{copular}$ may have been C/A *adj* missed	$V_{passive}$ may have been missed O she
627. ✔	Coal has been replaced by oil.	S coal $V_{copular}$ has been C/A *adj* replaced *PP* by oil	$S_{passive}$ oil $V_{passive}$ has replaced O coal
628. ✘	This difficulty can be avoided in several ways	S this difficulty $V_{copular}$ can be C/A *adj* avoided *PP* in several ways	$V_{passive}$ can be avoided O *iobj PP* in several ways O this difficulty
629. ✔	We are encouraged to go on with the project.	S we $V_{copular}$ are C/A *adj* encouraged *nom-cl* [*to-inf* V go O *PP* on with the project]	$V_{copular}$ are encouraged O *nom-cl* [*to-inf* V go O *PP* on with the project] O we

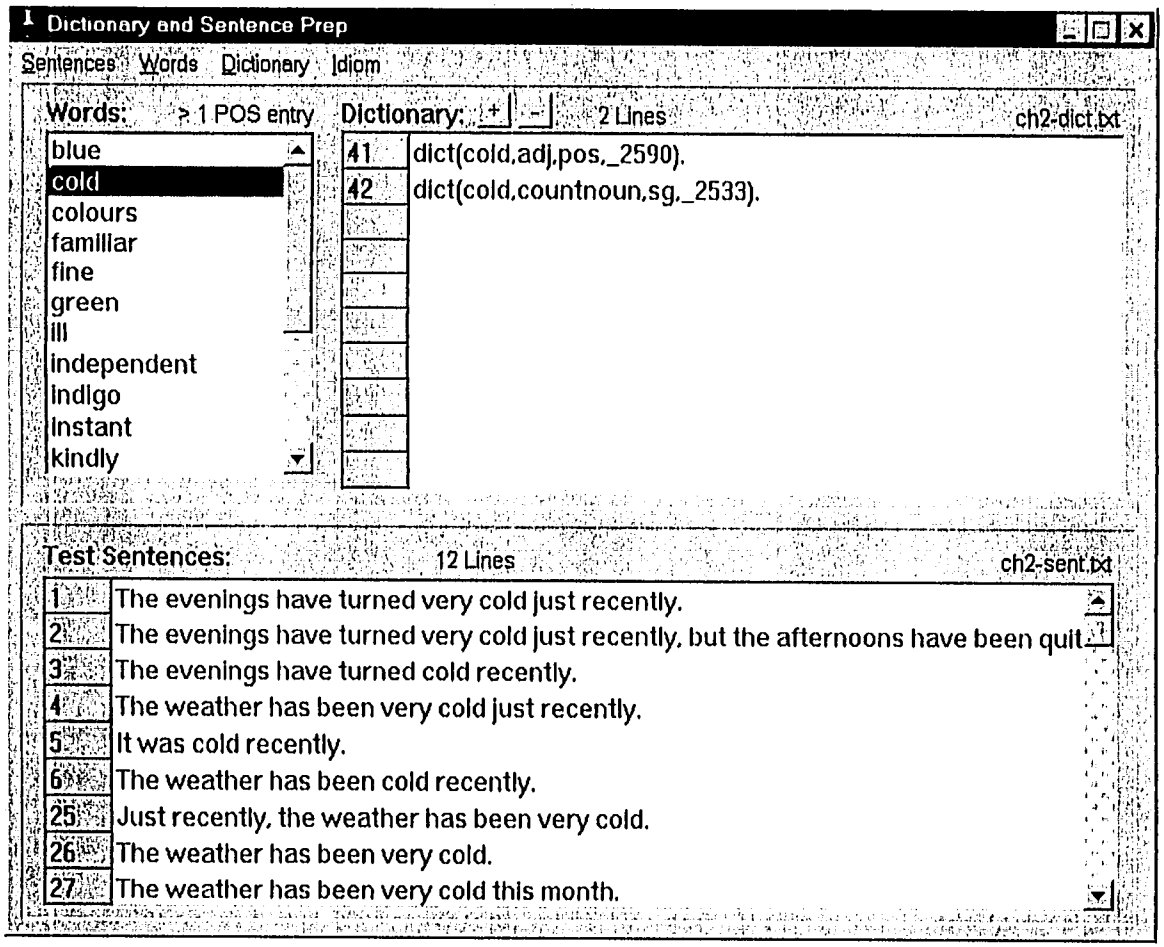
Appendix G: DICTMAN — DIPETT Dictionary Management Program User Interface

Appendix G shows examples of using DICTMAN to prepare sentence and dictionary files for parsing by DIPETT. The screen contains an edit window for the sentence file and the auxiliary dictionary file (or DIPETT's internal dictionary). The words entered into the dictionary file are listed in the Words list. Clicking on a word in the Word list causes the first occurrence of the word to be located in both the Dictionary and the Sentence edit windows.



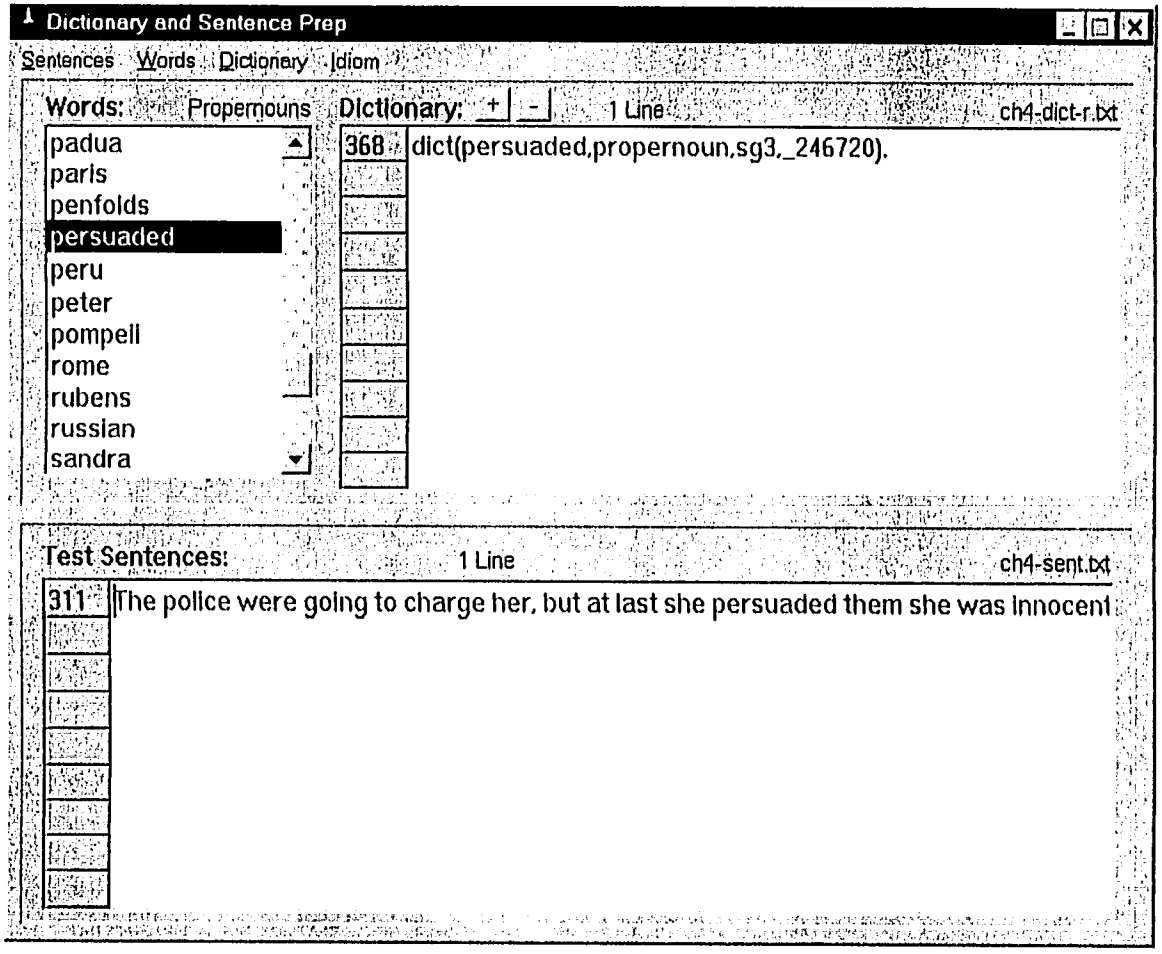
View Word

The view of the sentence and dictionary entries can be restricted to those that contain a particular word in the Words list. Clicking on a word in the Word list causes the dictionary entries and sentences that contain the selected word to be displayed. The entries that are displayed can be edited.



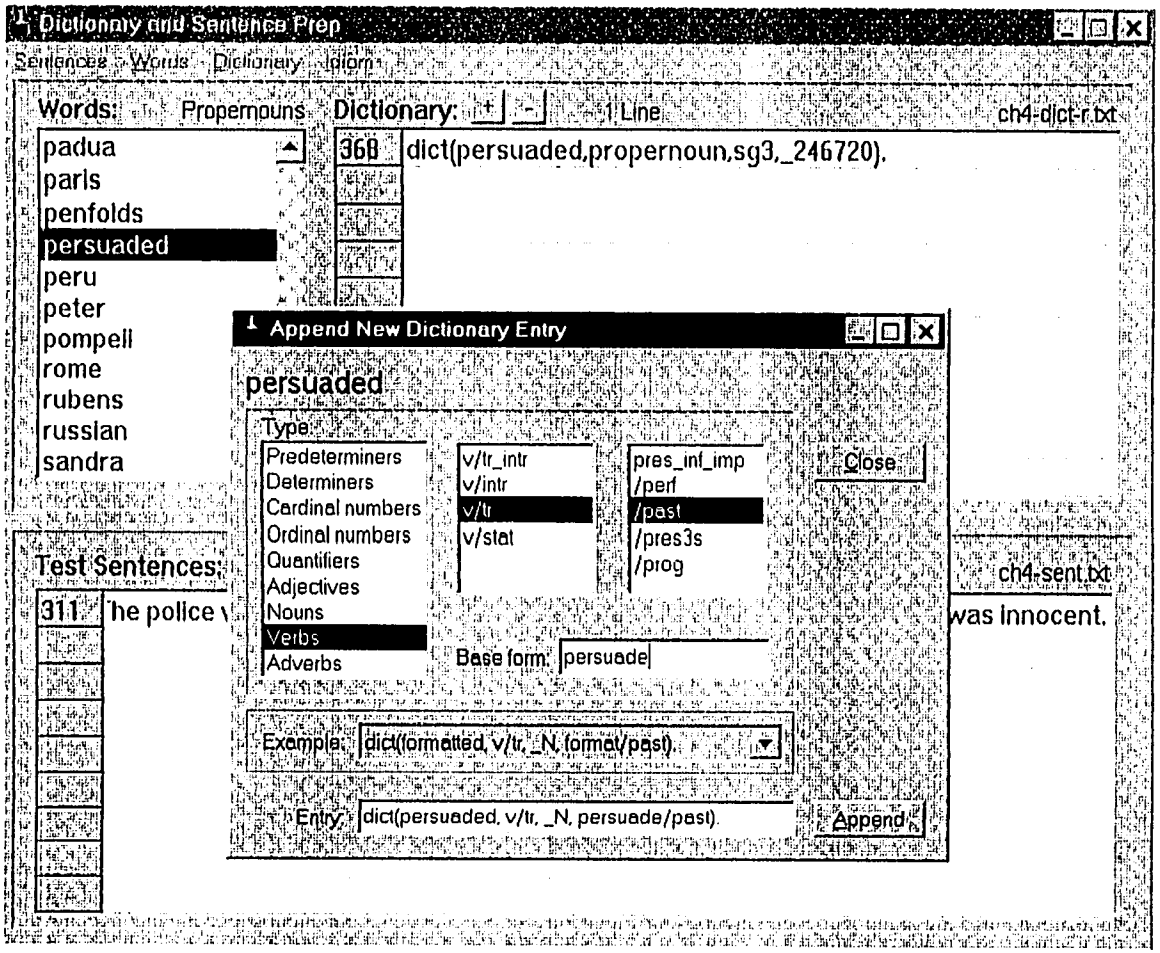
Viewing Particular Dictionary Entries

The Word list can show words that have multiple part of speech entries, words that are listed as both nouns and verbs, and words that are listed as proper nouns. The latter feature helps to identify words that are not known in Collins, as DIPETT assumes that they are proper nouns. Sometimes that assumption may be incorrect, as in this example where "persuaded" has been assumed by DIPETT to be a proper noun.



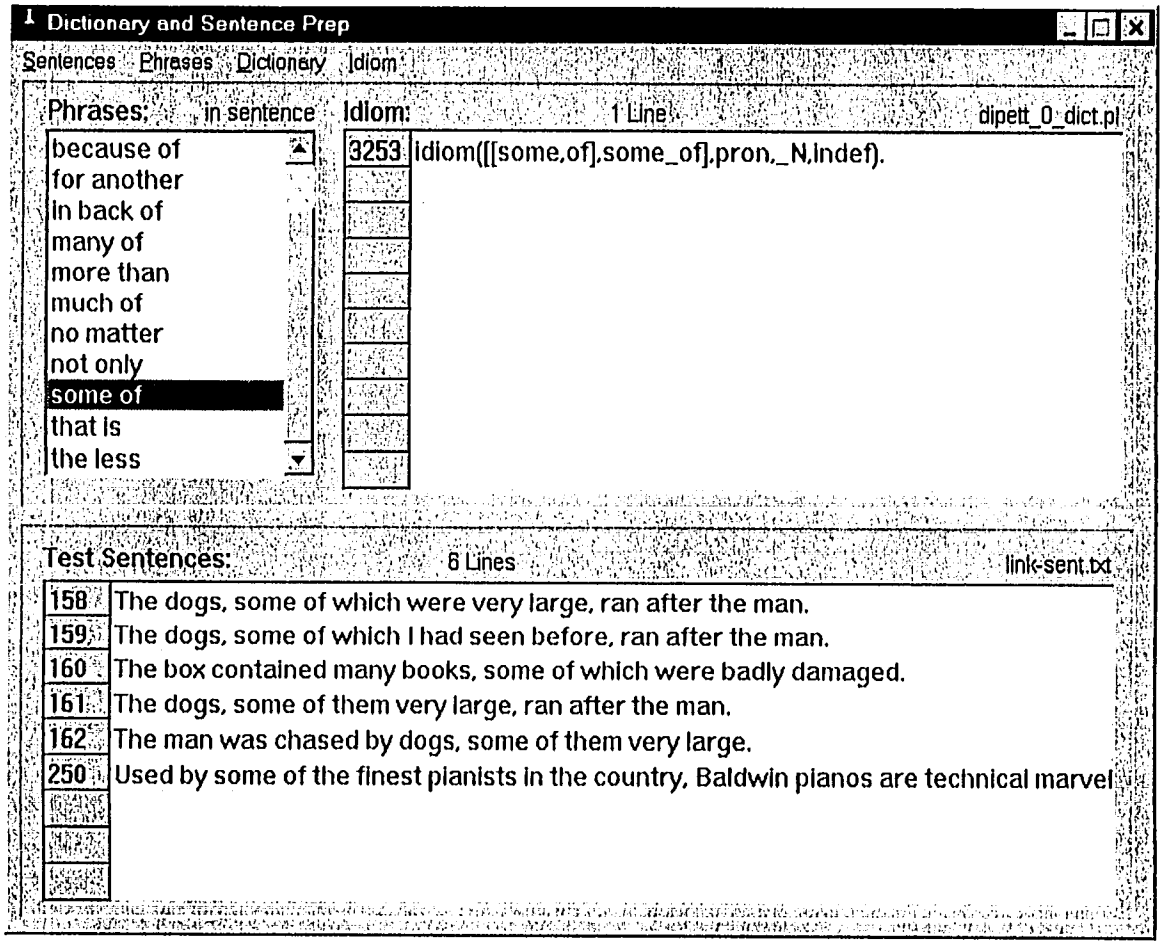
Dictionary Additions

Dictionary entries may be added or deleted by pressing the + and – buttons. DICTMAN knows the entry format and the options for all of DIPETT's part of speech categories.



View Idiom

DIPETT's internal dictionary file can be opened and displayed. All or any of the idiom expressions that are coded in DIPETT's internal dictionary can be located in the sentence file, if the idiom occurs in a sentence. The Phrase box may be set to contain all of DIPETT's idiom entries, or restricted to those that appear in at least one of the sentences in the sentence file. An idiom may have more than one entry in DIPETT's internal dictionary, and may appear in any of the test sentences.



Appendix H: Internet Addresses of Web-based Parsers

Appendix H shows Internet addresses of Web-based parsers visited.

Online Parsing Demonstrations

Parser	*Maintained by*	URL
Natural Language Processing Parser Demonstration Page	Georgetown University	http://www.georgetown.edu/cgi-bin/compling/slctscr.pl
Natural Language Parser Demo	Alpo Lind, Finland	http://pointti.vip.fi/
EngLite Parser — Functional Dependency Grammar Parser	Conexor, Helsinki	http://www.conexor.fi/analysers.html
ENGCG — Constraint Grammar Parser of English	Lingsoft, Helsinki	http://www.lingsoft.fi/cgi-pub/engcg
The Link Parser — A parser and grammar for English	Carnegie Mellon University	http://www.link.cs.cmu.edu/link/
Head-corner parser applied to Alvey Natural Language Tools grammar	University of Groningen	http://odur.let.rug.nl/~vannoord/CL97/
IPS — Interactive Parsing System	University of Geneva	http://latl.unige.ch/latl/latl_e.html
LINGO — Linguistic Grammars Online	Stanford	http://hpsg.stanford.edu/hpsg/lingo.html
PRINCIPAR — A Broad-coverage, Principle-based Parser	University of Manitoba	http://www.cs.umanitoba.ca/~lindek/principar.htm

Parsers Available for Downloading

Parser	*Maintained by*	URL
PC-PATR	Summer Institute of Linguistics	http://www.sil.org/pcpatr/
The PAPPI System: A Principle-Based Parser	NEC Research Institute	http://www.neci.nj.nec.com/homepages/sandiway/pappi/
Attribute-Logic Engine (ALE) System and Grammars	Universität Tübingen	http://www.sfs.nphil.uni-tuebingen.de/~gpenn/ale.html
Apple Pie Parser	New York University	http://cs.nyu.edu/cs/projects/proteus/app/