ANALYSIS OF A FINITE-STATE GRAMMAR FOR PARSING AVIATION-SAFETY REPORTS

by

César Dragunsky

B.Sc., Universidad Del Sur, 1991

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

In the School
of
Computing Science

SIMON FRASER UNIVERSITY
June 2004

APPROVAL

Name: César Dragunsky

Degree: Master of Science

Title of thesis: Analysis of a Finite-State Grammar for Parsing Aviation-Safety Reports

Examining Committee: Dr. Anoop Sarkar
Chair

Dr. Fred Popowich, Senior Supervisor

Dr. Oliver Schulte, Supervisor

Dr. Paul McFetridge, SFU Examiner

Date Approved: June 30, 2004

SIMON FRASER UNIVERSITY

Partial Copyright Licence

Abstract

With the growth of the World Wide Web in the nineties, alongside the increase in storage and processing capabilities of computer hardware, the problem of information overload resulted in an increased interest in finite-state techniques for Natural Language Analysis as an alternative to fragile, slower algorithms that would attempt to find complete parses for sentences based on general theories of language. As it turns out, shallow parsing, a set of robust parsing techniques based on finite state machines, provide incomplete yet very useful parses for unconstrained running text. The technique, however, will never provide 100% accuracy and requires that grammars be geared to the needs of particular data samples. In this project, we take a corpus of aviation safety reports parsed by Cass, an existing partial parser, with a particular given grammar, and look for instances of linguistic constructs whose treatment by the parser could be improved by modifications to the grammar. A few such constructs are discussed, and the grammar is edited to reflect the desired improvements. A parser accuracy measure is implemented and evaluated before and after the grammar modifications.

Acknowledgments

I probably cannot list in a single page all the people that I should thank for this.

First of all, my parents.

Second, people like Verónica Dahl and Diana Cuikerman, whose help when I was just arrived in Canada was invaluable and probably a big factor in the eventual realization of this project. My great friend Rick Ouellet shuold not be left without mention either.

I have to include all of the fantastic people at the School of Computing Science at SFU, all of whom have always been helpful and their help has many times been crucial in the concretion of this M.Sc. project.

Finally, my supervisors, Fred Popowich And Oliver Schulte, whose seemingly infinite patience, faith, and guidance, were the most important factor in making this project a reality. Along with them I want to thank the remaining members of the committee, Paul McFetridge and Anoop Sarkar, for their time and interest.

Contents

Approval ii

Abstract iii

Acknowledgments iv

Contents v

1 Introduction 1

1.1 Partial Parsing 2

1.2 Motivation for this Project 3

1.3 Improving Grammar Coverage 4

1.4 Measuring Parser Accuracy 5

1.5 Project Outline 6

2 Parser Accuracy Measures 8

2.1 Aspects of Performance 9

2.2 Some Existing Parser Evaluation Techniques 12

2.3 Important Parser Evaluation Projects 18

2.3.1 Parseval 19

2.3.2 Sparkle 24

2.3.3 EAGLES 25

2.4 Evaluating Cass in our Project: Discussion 27

3 Partial Parsing and the Cass Partial Parser 30

3.1 Notational Conventions 30

3.1.1 Notational Convention for Grammars . . . 31
3.1.2 Notational Convention for Parses . . . 31
3.2 Part-of-Speech Tagging . . . 33
3.3 Cascading Finite Automata . . . 33
3.4 Characterizing the Structures Built by Cass . . . 37
3.5 Cass Input and Output Files, their Format and Contents . . . 38
3.5.1 Mapping Standard Tagsets to Grammar-specific ones with `tagfixes` . 38
3.5.2 Compiling the Cass Grammar: `reg` . . . 39
3.5.3 Cass at Work: Inputs and Outputs . . . 41
3.5.4 Original Grammar Provided with the Scol Distribution . . . 41

4 The ASRS Database **43**
4.1 Origin of the Database Information . . . 43
4.2 Nature of the Database information . . . 44
4.3 Use of the Database Information . . . 45
4.4 Our Corpus . . . 47

5 Room for Improvement in the ASRS Database Corpus **49**
5.1 Treatment of the Forward Slash . . . 50
5.2 More on Punctuation: the Single Quote . . . 53
5.3 Further Punctuation Problems . . . 54
5.4 The Problem with 'away' . . . 55
5.5 The Problem with 'many' . . . 56
5.6 The Problem with 'when' . . . 56

6 Addressing the Problems **58**
6.1 Dealing with the Forward Slash Problem . . . 58
6.2 Dealing with the Single Quote Problem . . . 61
6.3 The Quotation Marks: a Problem too Complex to Solve Here . . . 62
6.4 Solving the Problem with 'away' . . . 65
6.5 Solving the Problem with 'many' . . . 68
6.6 Solving the Problem with 'when' as a Subordinating Conjunction . . . 72
6.7 Untested Fixes . . . 74

6.7.1 Numeric Ranges, Wind Expressions and Dates Containing Only Numbers 74

7 Evaluation: Experimental Results, Observations and Discussion **77**

8 Conclusion **84**

A Listing of the Test Corpus **87**

Bibliography **92**

Chapter 1

Introduction

Partial parsing techniques have proved useful in many real-world Natural Language Processing Applications that deal with unwieldy amounts of unstructured running text. Fast and efficient, they can provide impressive accuracy figures if their grammars adequately reflect the linguistic structures prevalent in the data.

A grammar that conveniently fits the data, however, is not trivial to implement. Coming up with it requires extensive analysis of a *corpus*, a representative sample of the data that the parser is expected to deal with. Grammar writing in this context is thus a very empirical task involving much trial and error.

In this project, we take an existing grammar for Cass (see [3]), a partial parser that can be freely downloaded and used for academic purposes. We look through a corpus of text from aviation incident reports parsed with this grammar and find a number of linguistic constructs whose treatment is not satisfactory. We propose changes to the original grammar that more adequately deal with such constructs.

To substantiate the claim that the modified grammar more properly fits the data than its predecessor, we implement the Parseval accuracy metric (see [5], [7]). This evaluation method produces four numeric figures which quantify aspects of the performance of the grammar under evaluation. By arguing that these quantitative indicators clearly suggest an improvement of the modified grammar with respect to the original one, we conclude that the changes implemented are indeed successful in yielding a grammar that more closely reflects the linguistic structures found in the input.

1.1 Partial Parsing

The idea of partial parsing is not new. As far back as the sixties, Zellig Harris ([14]) was proposing a full-fledged theoretical framework to analyze sentence structure by successive levels of finite-state machines. Another important historical precedent is the work of Kenneth Ward Church in his doctoral thesis ([9]). Here he focuses on the importance of finite-state techniques for natural language parsing as a means to achieve limited, albeit useful, structural analysis of running prose. The relative simplicity of finite-state-machines compared to the more expressive formalisms employed in obtaining more complete parses makes this method very appealing for real-world applications involving vast amounts of information. Hence there is a tradeoff between *competence* and *performance*: if we sacrifice some completeness and depth of analysis (competence), we will reap much needed gains in speed and memory requirements of parsers (performance).

With this in mind, an important line of research developed in the nineties, the beginning of the Internet era, powered by the need to make sense of large amount of unstructured data in what is known as the *Information Overload* problem. As is turns out, successful products were developed, one of them being the *Cass Partial Parser*, developed by Steven Paul Abney ([3]), which is available free of charge for academic research purposes in a package called *Scol*.

Partial parsing relies on a succession of filters being applied to the input, with the output from each being fed into the input of the next. Each of these filters consists of a finite-state machine, which can be denoted by a regular grammar. Cass allows the user to write their own grammars, in a language quite close to the standard regular expression notation used in the popular Unix regular expression language *regexp*, a de facto standard in many computational applications and platforms, and probably by far the single most popular notation for regular expressions.

Writing grammars for a partial parser should not be approached with the goal in mind of finding constituent labels of linguistic significance, but rather as a 'grammar programming' task. The focus is set on finding reliable delimiters of constituents so they can be isolated. The relations among constituents, such as attachment of prepositional phrases, are left unresolved. The resulting parse consists of shallow parse trees, where the main constituents are identified, their internal structure analyzed to the degree allowed by the limited information that the parser makes use of (i.e., only part-of-speech tags, no semantic

information).

These parses can be later used, for example, as a stepping stone for more involved analyses that do make use of a lexicon rich in semantic information. This would allow disambiguation of prepositional phrase attachment and structure of noun-noun sequences, among others. More interestingly, partial parsing taken as an first step in the analysis of prose opens the door to Natural Language Understanding tasks other than more complete parsing, such as Information Extraction. This is what makes partial parsing so appealing for applied Natural Language Processing.

In this project, we make use of the Cass partial parser and try to improve the coverage of a grammar that comes with it. This grammar was developed by Abney himself by a process of trial and error on corpus data. This grammar is to be considered a broad-coverage grammar of the English language, which must be tailored to fit the linguistic constructs prevalent in the writing style and type of language used in the data that it is expected to deal with for the particular application if high accuracy is desired.

1.2 Motivation for this Project

The ASRS (Aviation Safety Report System) is a program funded by the Federal Aviation Administration of the United States of America (FAA) and run by the National Aeronautics and Space Administration (NASA) (see [4]). The program collects, processes and stores reports of aviation-related incidents (with some limitations; for example, incidents involving criminal activity are within the jurisdiction of the FBI or other enforcement agencies and will not be allowed into the ASRS report database).

These reports are voluntarily submitted by aviation personnel who were involved in or witnessed an incident they consider worth reporting. The reports are confidential and any identifying information is removed from the text before input to the database.

The idea of the program is to generate advisories and warnings aimed at improving security in aviation working environments within the United States. It does this by gathering and analyzing input from all ranks of aviation industry workers, like pilots, flight assistants, mechanics, flight controllers, and so on.

This program presents an archetypical example of the problem of information overload. Because there are such enormous volumes of running, unstructured English text involved, finding and processing relevant information becomes challenging. Thousands of people must

be employed at high cost to perform largely mechanical text analysis tasks. Important information about flaws in aviation procedures might be buried deep within these texts, and hard to find unless a human being reads the passages detailing the danger and realizes their importance.

The tasks of the ASRS can be made much easier and less costly by computational technology whose first step is partial parsing of the texts involved. Removal of identifying elements in a report, for example, can be at least partially automated if proper names, places, times and the like can be identified by their structure. The reports can be searched much more effectively if relevant semantic concepts can be used rather than plain keywords. While partial parsing does not make use of any semantic information to compute its output, it does provide structures which semantically aware software can use to spot individual mentions of entities of conceptual significance, and infer relations between them concerning their roles in an incident. This would allow the report database to be searched for events, entities or relations among them, of more conceptual significance than a keyword search on unstructured text could provide. Statistics regarding the frequency of particular types of incidents, or the incidence of certain factors in them, would be possible to compute and lay out in visually appealing formats that would allow for at-a-glance interpretation.

But all of these useful computational tasks rely on accurate identification of constituents, which is why we set off, in this project, to bridge the gap between the grammar provided with Cass and the data found in the ASRS database.

1.3 Improving Grammar Coverage

To narrow the gap between the structures conferred by the original grammar and those found in corpus data, we first need to identify constructions which are not given proper structures. In this project, this was carried out by direct inspection of the original corpus by hand.

A few problems stood out right away: four easy-to-see problems with the handling of punctuation were found immediately, namely the handling of parentheses, quotation marks (`"`), the single quote (`'`), and the forward slash (`/`). As it turns out, the first two turned out to be harder to deal with than expected, and were left unsolved.

A few subtle sources of grammatical error were discovered later: in particular, the adjective *many* in phrases like 'many miles', post-head adverbial modifiers *away* and *apart*

in phrases like '10 miles away' or '100 feet apart', and the subordinating conjunction *when* in phrases like 'when I landed'.

The grammar was thoroughly inspected in search for the reasons for the mishandlings. The structure of the grammar needs to be well understood before any changes are considered, because there usually are non-obvious interactions between different parts of a grammar and between linguistic constructs in input text.

1.4 Measuring Parser Accuracy

If we hope to argue that our modifications of the existing grammar indeed constitute an improvement, we should provide some sort of quantitative argument to substantiate this claim.

For this purpose, we implement and run the Parseval parser evaluation metric. Parseval was developed in the early nineties [7], [5] and became a de facto standard mainly due to a series of very successful annual contests where authors were invited to test the performance of their parsing systems on different data. The gauge for parser accuracy in these contests was Parseval, which gave it enormous popularity throughout the nineties.

To compute this metric, a set of fifty sentences were picked from a corpus containing thousands of ASRS reports parsed by Cass with its original grammar. We call this our *pre-edition* corpus, in reference to the fact that it consists of parses obtained with the grammar before any modifications were introduced. The sentences were chosen so that each of them contains at least one word associated with a problematic linguistic construct, and trying to select a roughly equal amount of sentences containing each of these words.

These fifty sentences were also parsed by hand, carefully looking for 'correct' parses in every case – that is, parses that provided a decomposition of the sentences that would be useful for the kinds of applications that they are meant for. This hand-parsed set of fifty sentences is what we call our *golden standard* – our 'ultimate notion of correctness'.

After the proposed modifications were performed on the grammar, we used it to parse the same set of sentences with Cass. This corpus will be called the *post-edition* corpus by virtue of the fact that it reflects the changes introduced by edition to the grammar.

The metric rates the pre- and post-edition corpora by comparing each with the golden standard. Four numeric figures are produced: precision, recall, zero crossing brackets and mean crossing brackets. All of these measures are based on the notion of a bracketing,

understood as a pair of integers (i, j) stating that the grammar outputs a constituent in the sentence spanning position i through position j. Positions are counted at word boundaries. Position 0 is found immediately before the first word of the sentence, position 1 lies between the first and the second word, and so on until finally position n is after the last word of the sentence for an n-word sentence.

For a sentence of one of the pre- or post-edit corpora, a bracketing is said to be 'correct' if it is found in the golden standard. Constituent names are dropped, only the spans of the bracketings are considered. Precision for a sentence is defined as the number of correct bracketings found in the corpus over the total number of bracketings produced by the parse. Recall is defined as the number of correct bracketings over the total number of bracketings found in the parsed given to the sentence in the golden standard. A crossing bracket is defined as a bracketing that partially overlaps a bracketing in the golden standard. With this notion in mind, zero crossing brackets is the number of sentences in the corpus that have no crossing brackets, and mean crossing brackets is the total number of crossing brackets found in the corpus divided by the total number of sentences in the corpus.

These four numbers will allow us to argue whether our changes to the grammar are for the better or for the worse: an improved grammar should see the precision, recall and zero crossings go up, and the mean crossings go down, from its predecessor.

1.5 Project Outline

Chapter 2 of this project gives an overview of the field of parser evaluation. It discusses the motivation for research in this area, and classifies parser evaluation techniques according to a few different criteria. The criteria used by different authors are different, and in fact even partially overlapping. This chapter brings these together and bridges the gaps between classifications by relating similar notions across them and pointing out the differences between dissimilar ones. It ends by discussing the Parseval metric in detail, and dealing, one by one, with the many objections that experts have brought up against this measure technique over the past few years. As we argue, although very valid for the general case, these objections do not apply to our project and we adopt Parseval as a gauge due to its simplicity of both implementation and interpretation.

Chapter 3 discusses the most important concepts involved in partial parsing and how they apply in particular to our parser of choice, the Cass partial parser. It lays out some

useful notational conventions that are used throughout the text, and proceeds to define the notions of part-of-speech tagging and cascading finite automata, pivotal notions for this project. The implementations of these notions brought about in Cass are dealt with in detail, and finally the tools relevant to this project that come along with Cass in the Scol package are explained.

Chapter 4 delves into the ASRS program, its purpose, the origin of the information contained in it, its processing, and its nature. It ends with a section explaining our corpus and what the information found in it looks like.

Chapter 5 details, one by one, the problems we found with the original grammar in the corpus text. Chapter 6 goes into why we chose not to address some of the problems found, and how we chose to solve the rest of them. There is a final section on 'untested fixes', that is, modifications to the grammar that were perceived as beneficial for our purposes, but whose impact was not directly measured by picking sentences containing specifically those problems. They are, mostly, changes that complement the ones aimed at solving the problems identified above.

Chapter 7 shows the results of the implemented measures.

Chapter 8 wraps up the project by indicating what these numbers reflect about the success of our approach.

Chapter 2

Parser Accuracy Measures

The need for parser evaluation techniques arises on various fronts: they are used for monitoring the progress of ongoing projects, for assessing the adequacy of parsers for specific applications, as a purely theoretical gauge of a parser's coverage of the target language, as a means of comparison between two or more parsers, techniques or grammars, and more.

The undertaking of evaluating a parser's output, however, turns out to be riddled with difficulties. This is largely due to the inherent complexity of the task of natural language parsing, the great differences in parser output languages, and the diversity of the possible rationales for evaluation. Before speaking about particular methodologies, we will establish a taxonomy of parser evaluation methodologies. Authors have come up with different classifications according to different criteria. In Section 2.1 we will bring all these classifications together, determining a set of *dimensions* along which parser evaluation techniques can be classified. In Section 2.2 we will list and discuss a few of the most popular evaluation techniques and attempt to place each of them in one class along each of the dimensions introduced. It is important to mention here that these dimensions or levels of classification are not necessarily orthogonal, since they are independently developed attempts at bestowing some order upon a set of sometimes very tenuously related techniques. Although all the methodologies described below somehow assess the quality of a parser's output, they were devised in different frameworks and with different goals in mind, and bringing them together is not always straightforward.

2.1 Aspects of Performance

Carroll, Briscoe et al. ([7]) contains a representative list of parser evaluation methods, but does not provide a particularly elaborate taxonomy of them. In the classification provided in this paper, all parser evaluation methods fall into one of two categories: *corpus-based* and *non corpus-based.*

Corpus-based methods can be further subdivided into *annotated-corpus-based* and *unannotated-corpus-based.* We will call this dimension of classification the technique's *corpus requirement.* In measures that require an annotated corpus, the evaluation is usually based on an implicit assumption that all annotations in such corpus are 'correct'. This corresponds to the notion of *golden standard* defined in Section 1.1.

Other aspects of a metric, for example, are the ones discussed in Bangalore et al. [5]. In this paper, evaluation methodologies are separated into the following three categories:

- intrinsic,
- extrinsic,
- comparative

Intrinsic evaluation measures a system's performance in the context of the framework in which it was developed. Intrinsic evaluation picks a set of measurable aspects of parser performance, as defined in terms of the formalisms that define either the output language, or the computational process of parsing, or some other quantifiable notion related to the parser under evaluation.

Extrinsic measures are defined here as those which judge how satisfactory the system is for a particular task. Thus, they consider the parsing system in the context of an embedding application and how well the parser suits its purpose within it. A good example might be how many valid hits a search engine produces. What constitutes a 'valid' hit lies completely outside the realm of the parser, and this measure does not concern itself with any of the linguistic considerations involved in the creation or maintenance of the parser as a helper application, only with the direct usability of the results it produces.

In comparative measures, the focus is set on comparison. While intrinsic measures can be, and indeed are very effectively used in comparing, their comparative power is very restricted. They are, for example, widely used in comparing successive versions of a grammar [5]. Unfortunately, most of the commonly used intrinsic measures are only good for

comparing parsing systems whose output languages are very close.

To compare systems whose outputs are not that closely related, experts have devised metrics that would provide meaningful results even across computational representations of language structure. Bangalore et al. [5] and Carroll et al. [7], for example, provide their own proposals of evaluation frameworks which bridge the gaps between output representation languages, thus allowing for metrics that naturally allow for intrinsic and comparative measures, as well as some degree of extrinsic evaluation. An extensive discussion of these highly involved proposals is beyond the scope of this project.

The Human Language Technology Survey [11] (*HLT Survey* for short) provides yet another set of classifications. While Bangalore et al. focus on what the measurement hinges on (i.e., something that lies within the system, outside the system, or in another system we wish to compare), the classification provided here considers the purpose for which the measurement is taken.

The main division established here is also three-way. The three possible kinds of metrics are:

- adequacy evaluation,
- diagnostic evaluation, and
- performance evaluation.

Adequacy evaluation is akin to what was called *extrinsic* evaluation above. This publication also defines a criterion based on whether the system under evaluation is part of a larger system, or a system of its own.

In this context, *intrinsic* and *extrinsic* are defined as concepts similar to, but more general than, the ones found in [5].

'Intrinsic' is given here almost the same definition found in the cited paper by Bangalore et. al. If the subject of the evaluation is a subcomponent of the system, then intrinsic measures will evaluate it according to something internal to it, and extrinsic ones will do so in terms of its interactions with other modules of the system.

But if the subject of the evaluation is the whole system rather than a subcomponent, then this matches exactly what we called an intrinsic measure on our previous definition.

An extrinsic measure for a whole system would evaluate it in terms of something external to the parsing system, which, as mentioned above, the HLT Survey calls 'adequacy evaluation'. In this report we will use the terms 'adequacy evaluation' and 'extrinsic evaluation'

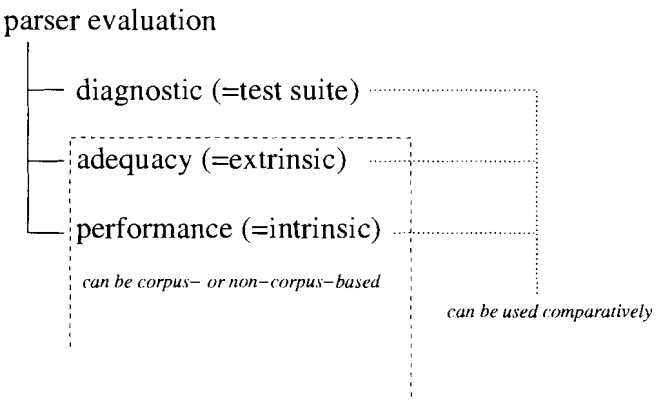

Figure 2.1: Classification of Parser Evaluation Techniques

interchangeably to denote whole-system extrinsic evaluation.

Performance evaluation "is a measurement of system performance in one or more specific areas". This definition is extremely general, since practically any metric could be argued to do this in one way or other, but it aims to capture a notion similar to that defined for intrinsic evaluation in [5], in which this category is defined as comprising those metrics which "evaluate the parser in the context of the framework in which it was developed". The HLT Survey further clarifies that these measures are used "to compare like with like, whether two alternative implementations of a technology, or successive generations of the same implementation". This implies a parallel with the aforementioned concept of intrinsic evaluation as defined in in [5].

Diagnostic evaluation, finally, is aimed at pointing out specific shortcomings of a system, usually one under development. As explained in [11], these metrics provide a profile of the system's performance "with respect to some taxonomization of the space of possible inputs". This clearly suggests test-suite-based methodologies, subclassified under the intrinsic category in [5]. Clearly a metric that is based on a list of test inputs each of which is thought to represent a particular grammatical construct (or set of constructs) which the parser is passible of failing upon encountering, is also likely to be devised "in the context of the framework in which [the parser] was developed" ([5]). Hence it makes sense to think of diagnostic measures as defined by [11] as a subset of intrinsic ones as defined by [5]. Figure 2.1 brings all these concepts together.

In the next section we will go through a list of some of the more documented and widely used parser performance metrics. While they can be used, at least to some degree, to assess

the adequacy of a parser for its task within a certain application, none of these measures are adequacy measures proper. Since the spectrum of possible embedding applications for a parser is unbounded, there is no real point discussing extrinsic measures in the absence of a concrete embedding application description.

It should also be kept in mind that these categories are not hard and fast: most techniques discussed can, for example, be used for intrinsic evaluation, which automatically makes them useful for comparison of systems with close enough output representations. Diagnosis and adequacy measures can be used for comparison too: the percentage of a test-suite that two different parsers cover, or the rate of successful retrieval provided by parsing systems embedded within the same application, can be used for comparison. And intrinsic measures can be used for diagnosis even if they are not based on an deliberately compiled test-suite, just like some facets of the system's internal working in some cases could be used to judge more or less directly upon its performance in an embedding application. Nonetheless, we will try to classify each technique as accurately as possible.

2.2 Some Existing Parser Evaluation Techniques

Both Carroll et al. [7] and Srinivas et al. [5] provide a survey of the most relevant existing parser evaluation techniques. Since the information they present is roughly the same, their summaries are combined here. The headings provided below are the names of the different techniques.

Linguistic Constructions Covered

An exhaustive list of the linguistic constructions covered by a parser, and a list of some not covered. The advantages of this measure are:

- no need for a corpus,
- easily obtained from a grammar,
- list of constructions not covered can be very useful in grammar development.

Its disadvantages are:

- the distinction between core and peripheral aspects of a construction is not clearly defined,
- much of the complexity can be in the interactions between constructions,
- hence, this measure alone does not provide a precise measure of data coverage.

This is the archetypical diagnostic measure, as it provides a tally of the constructions that are covered at the present stage of project development. A list of constructions not covered provides a "to-do list" for the grammarian. Of course the percentage of constructions covered, for example, can be used as an intrinsic (and hence also as a comparative) measure, and in the appropriate context this might even provide some insight into how well a parser satisfies the requirements of an application.

Coverage

Percentage of sentences from a corpus for which at least one parse is given. The advantages of this method are:

- no corpus annotation required,
- easy to compute even for large corpora.

Its disadvantages are:

- no guarantees on correctness of analyses. False positives may cause distortions in results,
- open to abuse. Grammars giving "flat" structures score high on this measure. A parser based on the grammar $S \rightarrow word*$ gives a perfect score.

This method requires a corpus, albeit an unannotated one will suffice. It is an intrinsic measure. It lends itself weakly to comparison, just like any intrinsic measure. The list of sentences for which no parse was found can be used as a diagnosis tool, too.

Parse Base/Average Parse Base

The parse base is defined as the geometric mean of the number of analyses divided by the number of input tokens for the sentence. In symbols, for a corpus $S = s_1, s_2, \cdots, s_n$, with a_i being number of analyses for s_i, t_i being the number of tokens for s_i,

$$PB(S) = (\prod_{i=1}^{n} \frac{a_i}{t_i})$$

This formula provides a measure of the ambiguity in the grammar for the particular corpus in use.

Observe that, in general, the number of parses for a sentence grows exponentially on its number of tokens, hence this measure underestimates for long sentences and overestimates for short ones. This motivated the definition of a related measure, dubbed average parse base, defined as follows:

$$APB(S) = (\sqrt[n]{\prod_{i=1}^{n} \sqrt[t_i]{a_i}})$$

Its advantages:

- easily computed
- succinct measure of the degree of ambiguity in a grammar

Its disadvantages:

- being just a measure of ambiguity, a low-coverage, unambiguous grammar does well,
- ambiguities in data and grammar interact, rendering this measure unable to provide a significant figure in comparing different grammars on different data

This method requires an annotated corpus. It is intrinsic.

Entropy/Perplexity

For a stochastic parser, these probabilistic figures give a measure of the degree to which a parser captures regularities in the corpus. The basic form of this measure works for one parser and one data set, but in conjunction with more elaborate methods it can be used to compare different training regimes on the same test data or the effectiveness of different language models. It can also be generalized to provide a model-independent measure of the inherent complexity of a corpus.

Its advantages are:

- has a clear interpretation in a probabilistic context,
- allows for the comparison of different language models on the same corpus.

Its disadvantages are:

- only applicable to probabilistic models
- provides only a weak measure of the accuracy of derivations, rather than an indication of the ambiguity of the model or the predictability of the data.

These two statistical methods depend on the existence of an annotated corpus, and can be considered intrinsic indicators of performance.

Part of Speech Assignment Accuracy

This can be measured as the ratio of correct tags per word, or by means of precision and recall.

Its advantage:

- availability of vast amounts of very satisfactory corpora.

Its disadvantage:

- nowadays most parsers take pretagged input, turning this measure into a gauge of the accuracy of the POS tagger rather than the parser itself.

This measure needs an annotated (POS tagged) corpus, and can be used as a performance indicator. POS tagging is discussed in Section 3.2.

Structural Consistency

The percentage of sentences in an annotated corpus that are consistent with the analysis provided by the corpus. Consistency is defined in terms of crossing brackets, which will be defined below in Section 2.3.1. This is a rather weak measure which recognizes only some forms of constituency conflicts.

Its advantage:

- it is stronger than full identity.

Its disadvantage:

- makes minimal use of the (expensive to produce) corpus annotations,
- it is open to abuse in the same way as Coverage; that is, flat structures obtain high scores.

This measure constitutes an example of relaxation of full identity as a comparison criterion between constituents with the aim of strengthening evaluation power. It is also a subset of the Parseval technique described below.

This is an intrinsic technique which can also be used as a comparison tool, and needs an annotated corpus.

Best-first/Ranked Consistency

In a probabilistic grammar, the percentage of the highest-ranked analyses that match (by full identity) a manually obtained annotation in a corpus. This measure can be extended in such a way that the first n parses are compared, with higher-ranked ones scoring higher.

Its advantages include:

- it provides a meaningful measure of how often a parser will deliver a correct parse,
- it measures the ambiguity of the grammar to some extent.

Its disadvantage:

- dependent on a corpus annotated with notation fully compatible with parser output.

This measure is annotated-corpus-based and intrinsic.

Tree Similarity

Several tree similarity measures have been defined, in an effort to relax full identity as a criterion for comparison of parser input with corpus annotations. The idea is to define a scale of how "close" the parse output by the grammar is to the golden standard.

The advantages of these measures are:

- more fine-grained than full identity
- tolerant to noise in corpus data

The main disadvantages of these measures are:

- what constitutes "closeness" is not clear, and ultimately is a matter of adequacy for a specific application,
- this family of metrics may sometimes turn out to measure a highly theoretical aspect of the data structures built by the parser with little significance for a real-world application.

This intrinsic performance measure demands an annotated corpus.

The Parseval Scheme

The GEIG (Grammar Evaluation Interest Group) or Parseval scheme is another relaxation of full identity which counts the number of matching bracketings between parser output and corpus annotation. Recall is defined as the ratio of this figure to the total number of bracketings in the corpus and precision is defined as the ratio to the total number of bracketings in the obtained parse.

Another metric usually bundled together with these two as part of this scheme is 'Crossing Brackets', a count of the number of bracketings in the obtained parse that do not properly contain or are not properly contained by any bracketing in the golden standard. This measure has been extended to make use of some constituent mark-up information. A more detailed account of this measure can be found in Section 2.3.1.

Its advantages:

- relatively simple corpus annotation required (for the variants using only bracketings),
- moderately fine-grained
- robust to corpus noise

Its disadvantages are:

- may penalize misattachments more than once,
- some misattachments may go unpenalized,
- it is not clear how the productions in the parser grammar correspond to the productions implicit in the corpus data, rendering the validity of the method somewhat dubious,
- penalizes analyses with more structure than the corpus, even if they are correct,
- cannot be applied to parsers which do not provide constituency-based analyses.

This is an intrinsic performance method based on annotated corpora.

Dependency Structure-based Scheme

To overcome the limitations posed by GEIG scheme's problems with matchings, sketched above as disadvantages of the method, this scheme is based on dependency structure. Constituency analyses are automatically converted to sets of dependency relationships.

This has the advantage that

- it makes use of dependency information that is not considered by many of the other metrics.

It has the disadvantage that

- some linguistic information that could be useful in later processing is lost in the transformation.

Another technique which exploits the idea of transforming constituent-based parses into sets of dependency relationships does so by flattening phrasal constituents into chunks which are later related by means of dependency relationships.

These methods depend on the existence of an annotated corpus. They are intrinsic performance metrics.

2.3 Important Parser Evaluation Projects

Parser evaluation is, as we have seen, not a straightforward task. Hence, it is natural that experts will gather in groups to carry out projects that attempt to define standards for

parser evaluation, in the hopes of evaluating real-world needs, finding methodologies that will address them, and setting said methodologies as standards among the potential infinity of possible choices so as to facilitate communication among groups of experts, a necessary condition for significant development of any technological field.

The first such project we will see here, Parseval, was developed in the early nineties and defined the very first de facto standard in parser evaluation [11]. Parseval is dealt with in Section 2.3.1.

The second one, Sparkle (see [6]), discussed in Section 2.3.2 is a major project whose main goal is the production of generic broad-coverage shallow parsers alongside with a framework on whose standards the creation of such software for a diversity of European languages is facilitated. The development of tools capable of learning the grammatical particulars of several European languages in the form of parametrisable platforms that can be trained for each of the different languages.

The third, EAGLES (see [13]), which we cursorily go over in Section 2.3.3, is entirely focussed on adequacy evaluation and thus not really within the scope of this project. It is included for completeness because of its relevance in the broad field of Natural Language Processing Systems Evaluation.

2.3.1 Parseval

Before 1991, when the Parseval metric for parser accuracy was developed, there was no objective and verifiable method for parser evaluation [11]. Parseval uses three figures to assess the performance of a parser: crossing brackets, recall and precision.

These three measures use bracketings as the basis for the comparisons that the measure comprises. Two bracketings are said to match if their spans, as well as their constituent names, are the same. Crossing brackets counts the number of constituents in the obtained parses that partially overlap constituents in the golden standard. That is, say we label each word boundary in a sentence, with label 0 naming the point in the sentence right before the first word, and label n denoting the point right after the n^{th} word in an n-word sentence. Then if there is a constituent in the golden standard which spans labels i through j, with $0 \leq i < j \leq n$, there will be a crossing bracket if the obtained parse has a constituent spanning positions k through l, where $k < i < l < j$ or $i < k < j < l$. In other words, there is no proper containment in either direction.

Precision counts the number of bracketings produced by the parser that match the golden

standard and divides this number by the total number of bracketings for the sentence in the obtained parse. The idea is to indicate how many of the bracketings produced are *valid*.

Recall is obtained by dividing the number of parser-produced bracketings matching the corpus and divides it, this time, by the total number of bracketings in the corpus. The idea is to indicate the proportion of bracketings in the golden standard that were indeed produced by the parser.

While this measure has great historical importance, there have been a number of objections over the years.

The Survey on the State of the Art in Human Language Technology [11], for example, explains that most of the information provided by the corpus is not taken into account, and that the level of agreement on details of linguistic description among the experts who created Parseval, and those who used the method after them, leaves much to be desired.

Bangalore et al. [5] describe the inconveniences in detail. For one thing, misattachments are penalized more than once. To illustrate this, they bring the following example: consider the sentence *She bought an incredibly expensive coat with gold buttons and fur lining at the store*. The following three alternative parses are given:

1.
```
[She
   [bought
     [[an incredibly expensive coat]
      [with [[gold buttons] and [fur lining]]]
    ]
     [at [the store]]
   ]
]
```

2.
```
[She
  [bought
    [[an incredibly expensive coat]
     [with [[gold buttons] and [[fur lining] [at [the store]]]]]
    ]
  ]
]
```

3.
```
[She
  [bought
    [an incredibly expensive coat]
    with
    [gold buttons]
    and
    [fur lining]
    [at [the store]]
  ]
]
```

The first sentence is meant for the golden standard, and reflects the common-sense interpretation that *at the store* modifies the act of the purchase. Thus the chunk is attached at the level same level as chunks *bought* and *an incredibly . . . and fur lining.* The bracketings found in this sentence are:

1. `[She ... store] (the whole sentence)`
2. `[bought ... store]`
3. `[an incredibly expensive coat with gold buttons and fur lining]`
4. `[an incredibly expensive coat]`
5. `[with gold buttons and fur lining]`
6. `[gold buttons and fur lining]`
7. `[gold buttons]`
8. `[fur lining]`
9. `[at the store]`
10. `[the store]`

The second sentence attaches *at the store* to the chunk *fur lining*, thus reflecting an interpretation in which the place **in the coat** where the fur lining is located is *the store.* The bracketings found in this sentence are:

1. [She ... store]
2. [bought ... store]
3. [an incredibly ... at the store]
4. [an incredibly expensive coat]
5. [with gold buttons and fur lining at the store]
6. [gold buttons and fur lining at the store]
7. [gold buttons]
8. [fur lining at the store]
9. [fur lining]
10. [at the store]
11. [the store]

The last sentence is simply a much "flattened" version of the first: here, chunks *bought*, *an incredibly expensive coat*, *with*, *gold buttons*, *and*, *fur lining* and *at the store* are all siblings. The bracketings found in this sentence are:

1. [She ... store]
2. [bought ... strore]
3. [an incredibly expensive coat]
4. [gold buttons]
5. [fur lining]
6. [at the store]
7. [the store]

It is arguable whether any real misattachments are happening here. Instead, this parse could be said not to state much of the information regarding the structure of the phrase `bought ... at the store`. Rather than state wrong structural information, it states none

at all.

However, the former sentence is much richer in information than the latter, and many a computational linguist would argue that a "good" measure should rank it higher.

What happens with Parseval's crossing brackets measure here is interesting: the sentence with the one wrong attachment, sentence number 2, has three crossing brackets:

- between `[[gold buttons] and [fur lining]]` and `[[fur lining] [at [the store]]]`
- between `[with [[gold buttons] and [fur lining]]]` and `[[gold buttons] and [[fur lining] [at [the store]]]]`
- between `[[an incredibly expensive coat] [with [[gold buttons] and [fur lining]]]]` and `[[an incredibly expensive coat] [with [[gold buttons] and [[fur lining] [at [the store]]]]]]`

The total number of bracketings for the first sentence is ten. Of the bracketings in the second sentence, the following six are correct (i.e., matched **exactly** by some bracketing in sentence 1): 1, 2, 4, 7, 9, and 10. Bracketings 3, 5, 6 and 8 are incorrect. This gives us a recall of $6/10 = 0.6$. The total number of bracketings in the second sentence is eleven, so in this case precision is $7/11 = 0.64$.

Sentence 3, on the other hand, has no crossing brackets. Its total number of bracketings is 7, and all bracketings are correct. This makes for a precision of $7/7 = 1$ and a recall of $7/10 = 0.7$.

The conclusion is clear: the less informative third parse, which 'risks less' by asserting less about the sentence, does better in every single score of the measure. The second parse is harshly penalized because of only one misattachment.

Another issue with this metric that they bring up in this paper is that additional structure that the parser might confer will be penalized: any bracketings that appear on a parse on top of those found in the golden standard will have a negative impact on the precision, because they will never match a bracketing found in the corpus.

The fact that Parseval had the format of an international competition, whose metrics were devised largely with the goal of allowing comparison among parsers with very diverse representations of linguistic information, was a significant factor in turning it into a de facto standard. In the last decade, however, the NLU community has been increasingly critical

of Parseval, as discussed above, and recent efforts (like [7], [8]), have tried to address these very valid concerns.

Carroll, Briscoe et al. [7] bring up similar objections to those in Bangalore et al. [5]. They mention that misattachments can be penalized several times, and that incorrect argument and adjunct identification can go unpunished in some cases.

Most other objections brought up in the literature and mentioned above concern the unpredictability of how the structures found in the corpus interact with the ones produced by parsers. This, however, is a general scenario for an arbitrary corpus and an arbitrary parser. As we will see below, this is not the case in our project, and Parseval turns out to be a reasonably fair gauge for the goals of this project.

2.3.2 Sparkle

Developed in the context of the data processing needs of the European Union, a multi-language information society, Sparkle seeks to address the most compelling issues in Natural Language Processing so as to set the stage for fruitful development of NLP applications in the EU.

According to the analysis of experts from several universities and private companies all over Eastern Europe who worked in this project, there were two most important needs to address:

- broad-coverage, shallow parsing techniques and tools, and
- lexical acquisition systems capable of learning aspects of knowledge from free text, in particular subcategorization, so as to build useful lexicons.

A natural consequence of setting these goals was the added need of evaluation techniques to assess the degree of success of the project upon completion.

For this reason, the Sparkle experts surveyed the existing parser evaluation techniques, which is reflected in Section 2.2 and they came up with their own scheme, which we briefly discuss below.

Since Sparkle comprised four groups working in developing parsers for different languages and based on different techniques, it is clear that their annotation scheme must be language-neutral and that no single annotation scheme will be a fair gauge for all types of parser involved. Thus, their annotation scheme involves three levels:

1. a *chunk* level scheme, in which sentences are broken up into chunks (see [1]), with each chunk having a grammatical category (e.g., nominal, verbal) and a head,

2. a *phrasal* level scheme, which specifies phrasal categories (e.g. noun phrase, verb phrase), boundaries, and hierarchical structure,

3. and a *grammatical relation* scheme, specifying dependency structure information.

In actual evaluation, at first only the English parser was evaluated at both the chunk and the phrasal level. All parsers were evaluated at the chunk level.

Sparkle's evaluation scheme features measures of recall and precision, and crossing brackets. For its recall and precision measures, Sparkle elaborates on the GEIG scheme presented above in Section 2.2, which computes recall and precision measures on an unlabeled bracket-based measure. Sparkle extends this notion to all of its levels of annotation by defining a match as exact identity of a chunk (brackets and label) at the chunk level, a constituent (brackets and label) at the phrasal level, or a relation at the grammatical level.

For its crossing-brackets measure, defined only at the phrasal level, two figures are defined: *mean crossings* and *zero crossings*. Mean crossings indicates the average number of crossing brackets per sentence and zero crossings indicates the percentage of sentences in the corpus that had no crossing brackets with the corpus whatsoever.

2.3.3 EAGLES

Running between February 1993 and May 1996, the EAGLES project involved over a hundred academic institutions and private companies, and constituted a major joint effort by the European Commission to develop standards for Natural Language Processing systems.

With the realization that any natural language system of practical use would entail a large project realized by teams comprising many individuals and hence requiring major cooperation, the idea that a serious effort towards the development of standards was the next step acquired enough momentum for the EAGLES project to become a reality.

EAGLES had five work groups, each respectively devoted to:

- computational lexicons,
- text corpora,
- computational linguistic formalisms,
- spoken language resources, and
- assessment and evaluation.

Each of the work groups has produced a series of recommendations in the form of reports, which can be freely downloaded from the EAGLES website or ftp server [13].

The last of the groups is the one whose work concerns us here. However, the EAGLES project's focus is on *adequacy*, or *extrinsic*, evaluation, see above. Thus the measures they propose, meant to assess how well a particular system fits a particular task, are not of very much use for us here. A discussion of the structure of the report, however, is appropriate.

The group has divided the kinds of systems considered into three categories:

- writer's aids,
- translator's aids, and
- knowledge management systems.

The report then consists of a main body and appendices on evaluating each of the above listed types of system. A few additional sections, more or less closely related to the main body, contain diverse information. The main body starts out by describing ISO 9126, an existing standard that was used as a starting point. Based on this, they attempt to build a *parameterized testbed*, namely a framework that will allow particular characteristics of each project to be evaluated to be provided as inputs to the framework.

The aspects of projects parameterized by this formalization are:

- the type of product to be evaluated,
- the type(s) of user of the product,
- descriptions of characteristics of systems that might be of interest to particular classes of users.

The output of the testbed will consist of a series of automated tests. Many tests are, however, impossible to completely automate due to their nature or limitations in the present

state of the art, and in these cases the framework will provide a series of instructions for a human tester on which test to conduct.

2.4 Evaluating Cass in our Project: Discussion

For the purposes of our project, which involves evaluation of the Cass partial parser, a measure based on crossing brackets, precision and recall, will be the most adequate. This is a consequence of several characteristics of our parser, data, and corpus annotations, as discussed below.

Firstly, crossing brackets, precision and recall provide easy-to-compute, objective, readily comprehensible and somewhat direct measures of the accuracy of a parser.

Secondly, the objections raised against these bracketing-based measures, while very valid and real concerns for the general case, are not applicable to this particular problem. In the succeeding paragraphs we will discuss the most relevant objections and refute their applicability to our project.

The first problem we will discuss, which was mentioned in Section 2.3.1, concerns mis-attachments. While it is true that a parser that attaches a constituent wrongly might have its output unduly penalized, further observation of the very nature of Cass output reveals that, because the parser precisely does NOT attempt to make many guesses on dependency information, it rather leaves all adjuncts unattached, or rather, attached to a node of that parse tree as low as possible, thus "containing" ambiguity without making any risky guesses.

Since the test corpus will be created *ad hoc* for this project, this particular characteristic of Cass output will be kept in mind, and no dependency information will be encoded into the corpus, only the kind of constituent information that Cass provides. The measurements obtained from this work are meant to be used in applications where shallow parsing is the kind of information sought after, so we are not really interested in measuring accuracy of attachment information.

So for example, if we had a sentence like

I saw the man on the hill with the telescope.

and we know that the correct interpretation attaches the prepositional phrase *with the telescope* to *the hill* (e.g., *the hill with the telescope* is an astronomic observatory of some

sort), we would not use in our golden standard a parse like this:

```
[#c [#c0 I saw]
    [#nx the man]
    [#pp on [#nx the hill [#pp with the telescope]]]]
```

but rather

```
[#c0 [#c0 I saw]
     [#nx the man]
     [#pp on the hill]
     [#pp with the telescope]]
```

That is, will deliberately ignore attachment information, even when it is known to us, to make the golden standard look like the analysis we would expect a partial parser to come up with, rather than a complete analysis with all attachments resolved.

Other objections claim that bracket notation does not allow for disconnected trees, since root attachment is the shallowest that can be represented. This would imply, the argument follows, that metrics based on bracket containment are inadequate for shallow parser outputs. Once again, it is the fact that this is known in advance that makes this argument inapplicable to our case. Since the corpus will be hand-coded by us, this fact will be kept in mind and as a result no incompatibilities will be found between the golden standard and parser output. In other words, neither the parser nor the golden standard really distinguish root attachment from non-attachment, so no "disagreements" could possibly arise. Additionally, and once again, dependency structure is not the kind of information sought by the prospective applications of these measurements, and our corpus annotations will reflect this in their shallow structure. Hence no distortions of the measurement will be caused by this apparent flaw.

Parseval has been used without constituent names in the past, but implementations of the measure have predominantly only considered two bracketings to match only both their spans and their constituent names did. For this project, however, we have chosen to drop the constituents names. This allowed us to simplify enormously the implementation of the measure. We will argue that there is no substantial loss in the validity of the metric due to this decision.

On the one hand, the idea of partial parsing is that of providing "islands of certainty" ([3]), that is, to identify the boundaries of relevant linguistic constructs as accurately as

possible. Towards this goal, finite-state grammars make use of many symbols which represent reliable boundary delimiters, rather than meaningful linguistic information. The fact that symbols in a finite-state grammar do not necessarily have linguistic significance puts their usefulness as constituent identifiers in question.

Related to the above is the fact that potential applications of this project are most likely going to be used for applications performing Information Extraction tasks. One good example of this would be finding all the explicit references to names of people involved in an incident to have them automatically replaced or removed prior to entry to the database. This tells us that there is likely to be a whole layer of applications running on the output of the parser, likely semantically aware applications which are much better equipped to choose linguistically meaningful constituent labels for the phrase boundaries identified by Cass.

Another relevant point is that in this project we are comparing two minor variations of the same grammar. In this light, the grammatical symbols involved in both versions should not be expected to differ significantly. Adding constituent names would have made the implementation of the measure much more complicated and probably would not have had a visible impact in the final values.

In conclusion, we will adopt a simplified version of the Parseval metric for this project, since it appears to be the simplest to implement and interpret, and the objections raised in the literature do not apply to our case.

Chapter 3

Partial Parsing and the Cass Partial Parser

The Cass partial parser, written by Steven Paul Abney, is a fast, reliable partial parser suited for information retrieval and other applications where a complete analysis of the relations among constituents in a sentence is not as relevant as the identification of the constituents themselves.

Cass works based on a cascade of finite-state machines. Hence there is no backtracking, which makes this parser much faster than those based on more traditional, non-deterministic (yet more complete) syntactic analysis techniques.

This chapter starts by defining notational conventions that will be used throughout this report in Section 3.1. It proceeds to defines the notion of POS tagging, basic to this project, in Section 3.2.

3.1 Notational Conventions

Before we proceed to explain how Cass works, we need to define some notational conventions that will be used in this section and throughout the report. We also present some background information on part-of-speech (POS) tagging, a step in the parsing process usually carried out before parsing. This project assumes that all its input has already undergone this process.

3.1.1 Notational Convention for Grammars

For grammars, we list one production per row. Grammars are structured by levels, see Section 3.3. Every level is introduced by a semicolon followed by the name of the level, in a line of its own. The levels are listed from lower to higher: the filter corresponding to the first level from the top is applied first, then the remaining ones in the order in which they occur. All productions belong to the last level declared.

Below is an example grammar that we will use in Section 3.3:

```
:chunk
  NP -> D? N+;
  VP -> V-tns | Aux V-ing;


:pp
  PP -> P NP;


:clause
  S -> PP* NP PP* VP PP*;
```

The notation is pretty much the standard for regular expressions: `X?` matches 0 or 1 occurrences of `X`, `X+` matches 1 or more occurrences of `X`, `X*` matches 0 or more occurrences of `X`, the upright bar (|) denotes union, juxtaposition denotes concatenation. All rules end with a semicolon. The first two productions belong to the first level, `:chunk`. The third belongs to the `:pp` level and the fourth one belongs to the last level, `:clause`.

3.1.2 Notational Convention for Parses

The notation we will use for parses uses bracket containment to denote ancestry in the parse tree. I.e., a constituent occurring immediately within another is its child in the parse tree. Grammatical categories are noted by an identifier starting with the symbol "`#`". Trees are shown only to the depth that is relevant to the discussion to avoid unnecessary cluttering. POS tags are also only shown when relevant. For the same reason, levels of the tree are marked up by listing them on separate rows and using visually appealing indentation. For example, the parse tree shown in Figure 3.1 would be noted in this report as shown in Figure 3.2. A tree for the same parse is shown abbreviated in Figure 3.3, with its text

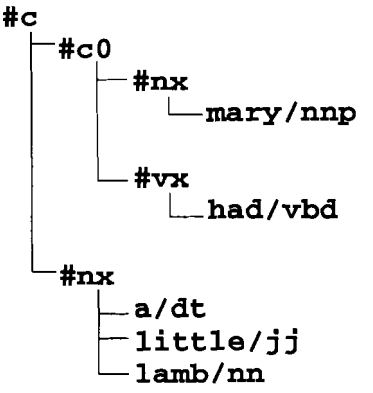

Figure 3.1: An Example Parse Tree

```
[#c
  [#c0
    [#nx mary/nnp]
    [#vx had/vbd]]
  [#nx a/dt little/jj lamb/nn]]
```

Figure 3.2: Text Notation for Parse Tree in Figure 3.1

counterpart shown in Figure 3.4. Notice how in the abbreviated tree the structure of the `#c0` is flattened out, and how all POS tags are lost, but for the tag for *little*. This type of abbreviation would be used in a context where the structure of the `#c0` is not relevant and the tag for the adjective is of interest for some reason.

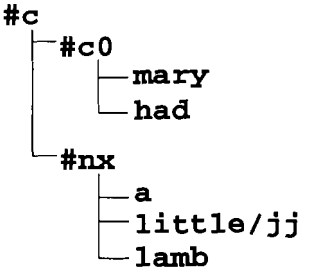

Figure 3.3: Abbreviation of Parse Tree from Figure 3.1

```
[#c
  [#c0
    [#nx mary had]]
  [#nx a little/jj lamb]]
```

Figure 3.4: Text Notation for Parse Tree in Figure 3.3

3.2 Part-of-Speech Tagging

Part-of-Speech, or POS, tagging is the process of labeling each word in a sentence with the one tag from a pre-established set or *tagset* that best describes its syntactic role given its local context. There are several standards that define these pre-established sets. Among the most popular are (from [15])

- the Brown Corpus Tagset,
- the Penn Treebank Tagset,
- the C5 tagset used by the Lancaster UCREL project's CLAWS tagger [19], and
- the related, larger C7 corpus (also from [19]).

Table 3.1 taken from [15], shows the Penn Treebank Tagset, which is the one that the original grammar's tagfixes file assumes at its input [3].

In this report, we will indicate tags by typing them in monospace font and preceding them with a forward slash. For example, `/in` refers to the tag for prepositions and conjunctions. All tags mentioned in this document refer to their meanings in Table 3.1.

3.3 Cascading Finite Automata

Cascading finite automata dates as far back as the sixties, when Zellig Harris [14] devised a technique based on successive passes of sentences through finite-state machines. In his work, he provides a full-fledged theoretical framework and extensively analyzes the properties and limitations of his approach as a theory of language.

The Cass Partial Parser, introduced in Section 1.1, is the parser in use for this project and it is an implementation of finite-state cascades. The approach using cascading finite-state machines to parse a sentence passes it through a series of "filters", each comprising

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign Word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive Wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar Sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound Sign	*#*
PDT	Predeterminer	*all, both*	“	Left Quote	‘ or “
POS	Possessive ending	*'s*	”	Right Quote	’ or ”
PP	Personal Pronoun	*I, you, he*	(	Left parenthesis	[, (, {, <
PP$	Possessive Pronoun	*your, one's*	)	Right parenthesis	],), }, >
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	. ! ?
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	: ; ... – -
RP	Particle	*up, off*			

Table 3.1: the Penn Treebank part-of-speech tags (with punctuation)

a finite-state automaton. The first filters identify small, nonrecursive structures or *chunks*, like NPs, VPs, APs, AdvPs and the like. Subsequent layers of finite-state machines confer structure in terms of the constituents output by previous levels.

Any sequences not recognized by a finite-state automaton will not be rejected: rather, they will be punted[1] to the output, that is, fed to the input of the next-level automaton in the position they appear in the original sentence without any further modifications or annotations.

The idea behind this is that higher-level filters might find a pattern involving the unrecognized strings and the constituents found by the levels the sentence has passed through so far. Cass thus bulids on the idea of "islands of certainty": whenever you see a recognizable

[1]The term *punt* here comes from american football, where it means "to abandon the current attempt to score and kick the ball away" ([2]).

pattern, identify it immediately, ignoring whatever cannot be dealt with. This way, the parser can give a "best effort" parse consisting of partial derivation trees even if it cannot find a structure for the whole sentence.

To illustrate the above explained ideas, here is an example with three levels of automata, their corresponding grammars and how they incrementally come up with a parse tree for an example sentence (from [3]). The example grammar is the same we used in Section 3.1, shown here again for ease of reading:

```
:chunk
  np -> d? n+;
  vp -> v-tns | aux v-ing;

:pp
  pp -> p np;

:clause
  s -> p* np pp* vp pp*;
```

Let us see how Cass would go about parsing the following sentence with this toy grammar:

The woman in the lab coat thought you were sleeping

The first step is part-of-speech tagging: Cass assumes its input is POS-tagged. There are a number of products which perform a satisfactorily accurate job of tagging words with their corresponding part of speech. Assuming a simple, 'toy' tagset in which `/D` is the tag for determiners, `/N` is the one nouns, `/P` is for prepositions, `/V-tns` for transitive verbs, `/Aux` for auxiliaries, and `/V-ing` is for verbs in the present participle, the correct tagging for this

sentence would be:

```
the             D
woman           N
in              P
the             D
lab             N
coat            N
thouhgt         V-tns
you             N
were            Aux
sleeping        V-ing
```

By following the chunk-level grammar, we find the following structural annotation for the sentence:

```
[#np the woman]
in
[#np the lab coat]
[#vp thought]
[#np you]
[#vp were sleeping]
```

Notice that an ambiguity arises here: both *the lab* and *the lab coat* match the right side of the rule `NP-> D? N+`.

In this case, the ambiguity is resolved by taking the longer phrase, namely *the lab coat.*

This output from the first-level automaton will be fed to the second level of filters, which will render the following structure:

```
[#np the woman]
[#pp in [#np the lab coat]]
[#vp thought]
[#np you]
[#vp were sleeping]
```

By virtue of the only rule in this level, the preposition and the succeeding NP are brought together in a PP. The next level transforms this to

```
[#s [#nx the woman]
    [#pp in [#np the lab coat]]
    [#vp thought]]
[#s [#nx you]
    [#vp were sleeping]]
```

This shows a fundamental characteristic of Cass output that will be discussed in the next section: the subordinate clause is not contained within its containing clause, but simply juxtaposed.

3.4 Characterizing the Structures Built by Cass

Because it is based on finite-state automata, Cass does not allow for true recursion. All the grammars shown in the above three-level example are regular; this is not a coincidence. Clearly, since each level is a finite-state machine, any grammar that denotes it must be regular. Therefore, true recursion is not allowed: the right sides of rules can contain only symbols from the left sides of lower levels, never from higher ones. To be completely strict, recursion is actually allowed, but only in the very limited form of tail recursion, like in regular grammars. Nested structures are thus replaced in Cass by sequential structures, as shown in the example, where a subordinate clause occurs by the side of its containing clause rather than within it.

The same happens with other constructs, like noun-noun modifiers, where the internal structure of the ensuing NP is not analyzed. The whole phrase is simply rendered as a list of nouns constituting a noun phrase.

This characteristic of Cass output makes for what is called "containment of ambiguity": local ambiguities are kept local, without engaging the parser in a global analysis involving all possible local structures. slowing down the parsing process as happens in traditional CFG-based parsers (and by the way, in most cases not even really producing a realiable result, given the involved semantic and pragmatic considerations that usually enter into structural ambiguity resolution).

Cass never rejects a sentence: in the worst case, a sentence to which no grammatical structure can be ascribed will look like a flat list of the words that it comprises with their corresponding POS tags. What usually happens is that parts of the sentence can be analyzed and others cannot. In this case, as mentioned above, the output of the algorithm will show "islands of certainty", that is, the substrings of the whole sentence that can be parsed will be parsed, with everything else simply "punted" at the top level in the output tree.

3.5 Cass Input and Output Files, their Format and Contents

Cass was distributed by Steven Paul Abney in a package called Scol. In this section we will discuss the particular utilities in Scol that are relevant to this project, with their inputs, outputs and a brief description of their tasks. All the explanations found here were taken from [3].

3.5.1 Mapping Standard Tagsets to Grammar-specific ones with `tagfixes`

Because POS tags are inevitably embedded in any grammar, where they act as terminal symbols, tagsets are always grammar-dependent to some degree. This is why Scol provides a means to map tagsets to more grammar-specific ones if required.

This mechanism is granted by a program, `tagfixes`, which operates in one of two possible modes, each activated by either of the switches `-c` and `-f`, as described below.

When given the `-c` switch on command-line invocation, `tagfixes` works in *compile* mode. It takes as input a `.fx` file and compiles it into a `.fxc` file. The input `.fx` file has the following format: every line of the file consists of three tab-separated fields specifying a single mapping. Thus, a line reading *"x y z"* means "on finding word y with tag z, change the tag to x".

It is possible to use an asterisk (`*`) in the second or third fields to mean any word or tag, respectively. So for example, the line

```
nnp     rocky   *
```

means "on Finding the word `rocky`, give it the tag `nnp` no matter how it is tagged to start with", and a line saying

```
noun    *       nn
```

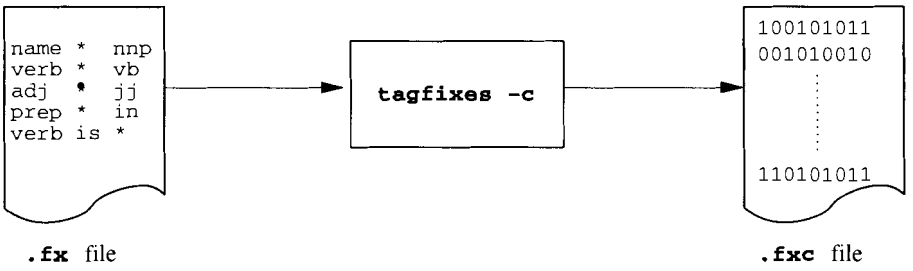

Figure 3.5: The `tagfixes` File Complilation Process

signifies "map all occurrences of the tag `nn` to `noun`, no matter the word".

The output `.fxc` file contains the same information as the `.fx` file but is in a format that allows for more efficient use of it by `tagfixes` in its other mode of operation, discussed below.

The whole process of grammar compilation is illustrated in Figure 3.5.

When given the switch `-f`, `tagfixes` takes as inputs

- the name of a `.fxc` file,
- the name of a file containing POS-tagged text (usually given the extension `.tag`).

This will return the text contained in the original `.tag` file, with the tags changed as per the rules listed in the `.fx` file.

The file consists of two kinds of lines: markup lines and word lines. A markup line is a line containing no tabs; their actual contents are ignored by both `tagfixes` and Cass. They are interpreted by the latter as sentence delimiters. Word lines contain a word and its POS tag, in that order, separated by tabs. As mentioned above, spaces are allowed within both words and tags by the rule that considers tabs as the only valid field separator. This process is illustrated in Figure 3.6.

In both the `.fx` and the `.tag` files, whitespace other than tabs is significant and a string containing spaces is treated as a single string. The spaces are considered part of the identifier.

3.5.2 Compiling the Cass Grammar: `reg`

One of the inputs to Cass, as discussed in Section 3.5.3, is a grammar. The Cass grammar format is discussed in Section 3.3.

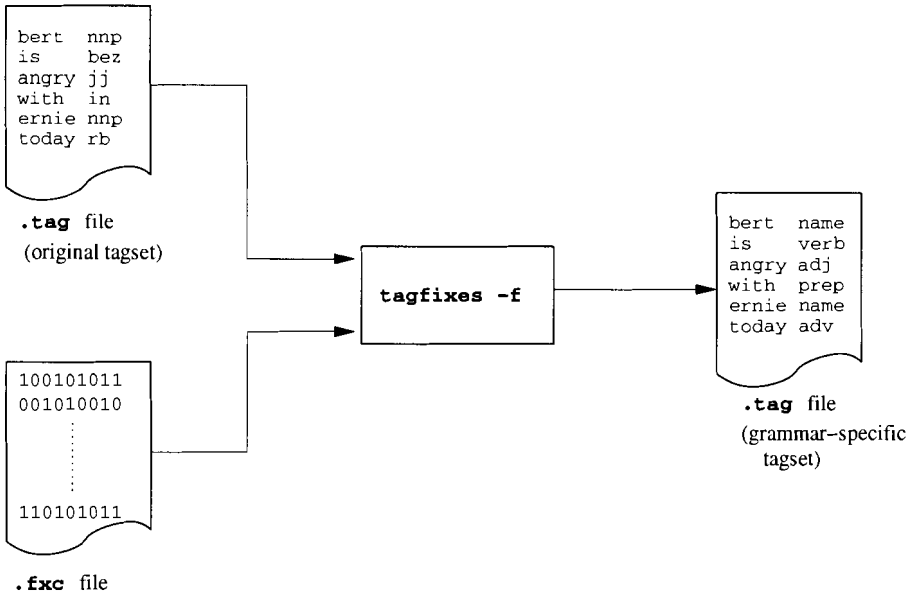

Figure 3.6: The `tagfixes` Mapping Process

For Cass to be able to use this grammar, it needs to have it compiled into a format that it can use at run-time, called the *Finite State Cascade* (`.fsc`) format. The program `reg` takes as input a grammar in a textfile (normally with extension `.reg`), and compiles it into a `.fsc` file. The `.fsc` file name is then used as one of the input arguments upon invocation of Cass, as explained in Section 3.5.3. The process of grammar compilation is illustrated in Figure 3.7.

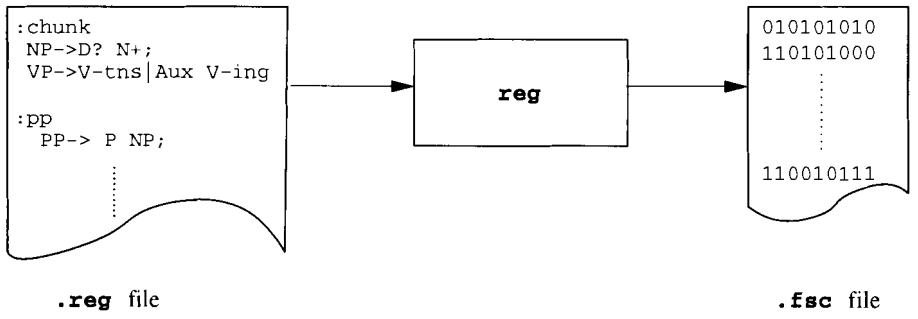

Figure 3.7: The Grammar Compilation Process

3.5.3 Cass at Work: Inputs and Outputs

The program that contains the code of the Cass Partial Parser itself is in a file called `cass`. When invoked from the command line, `cass` takes the following arguments:

- the name of a Finite State Cascade file (`.fsc`),
- the name of a file containing tagged text (`.tag`).

Cass's output consists of text describing the parse that Cass finds for the given `.tag` file using the grammar encoded in the given `.fsc` file. Any tagfixes desired must be done prior to use of Cass.

Cass provides a choice of three output formats. The only one dicussed here is the *forest* format, which is Cass's default and the one we use for this project. The forest format consists of standard parse trees in a notation similar to that used in Section 3.4.

Here is an example Unix-style command-line invocation of `cass`, with tagifxes included:

```
tagfixes -f h1.fxc text1.tag | cass -g h1.fsc
```

The `tagfixes` invocation takes a compiled mapping list in `h1.fxc` (which should have been previously compiled by using the `-c` switch, see Section 3.5.1) and writes a copy of contents of `text1.tag` with the corresponding tag transformations applied into the standard output. The `cass` process takes it from there and writes into the standard output the parse it obtains for it by using the grammar in `h1.fsc`. The `-g` flag tells `cass` to use this file as its grammar. This is illustrated in Figure 3.8

3.5.4 Original Grammar Provided with the Scol Distribution

Scol comes with a few example grammars and tagfixes files. Of those, the most complex ones, meant for full-scale broad-coverage parsing of english text, are the grammar `e8.reg` and the tagfixes file `e8.fx`. These files were written by Steven Paul Abney himself [3] based on empirical observations of corpus data.

Our project takes this grammar and tagfixes file as a starting point for improvement. All the sentences found in the ASRS sample corpus (see Section 4.4) are parsed using these. Our research suggests, implements and evaluates modifications of these files. Improvement is judged taking the Parseval (see Section 2.4) results for the aforementioned sample corpus as a baseline.

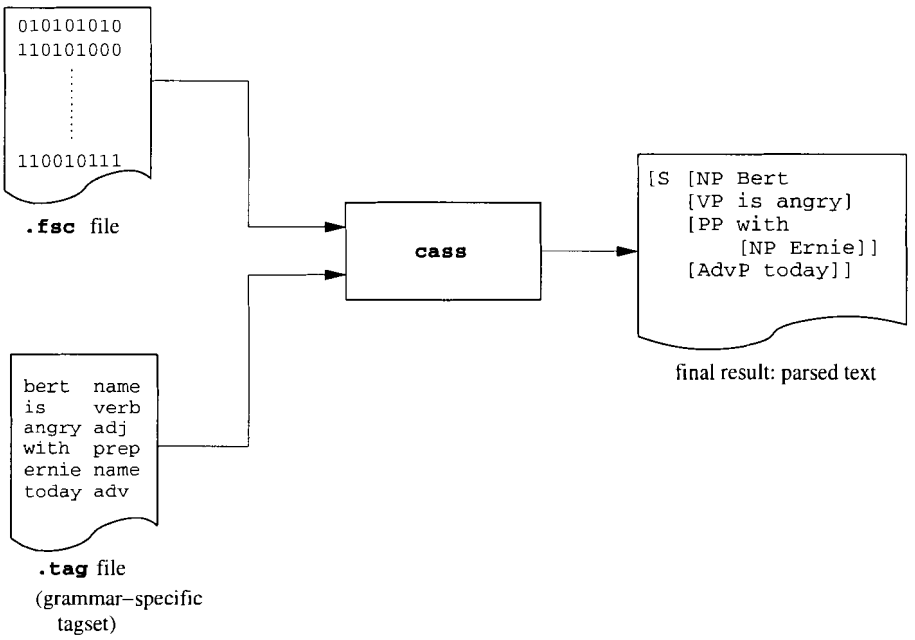

Figure 3.8: The Cass Partial Parser's Inputs and Outputs

Chapter 4

The ASRS Database

We will be applying the Cass partial parser, which was introduced in the previous chapter, to a selection of Aviation Safety Reports. Recall that the ASRS (Aviation Security Report System) is a voluntary, confidential aviation safety incident report system funded mostly by the FAA and run by NASA (see [4]).

The first step of the report processing is the search for situations towards which undelayed attention of competent authorities is required. When found, the ASRS issues "alerts" aimed at immediate correction of the offending situation. This is one of the outputs of the program.

Other outputs of the program include topical research on specific safety issues concerning the aviation community, two periodical publications for distribution among aviation personnel, and operational support by means of interaction with other organizations.

4.1 Origin of the Database Information

Pilots, crew members, air traffic controllers, and other aviation personnel voluntarily report incidents that compromise aviation safety. Personnel are encouraged to report incidents that they witness or participate in by a confidentiality warranty and some minor penalty waivers. Accidents and incidents involving criminal activity are not contemplated by this warranty and should not be reported to ASRS.

These voluntary reports are examined by no less than two aviation experts. ASRS counts on its corps of experienced pilots and air traffic controllers for this task.

4.2 Nature of the Database information

The one product of the ASRS program that concerns us in this project is the ASRS database. Each record of the database consists of two parts. The first part is a highly structured account of some aspects of the incident. It is in the form of a feature structure tree, with entries for time and place, for aircrafts, people, events, environment (i.e., weather conditions and the like), components (of planes, specially if something fails or is damaged in the incident), factors (facts or situations that might have contributed to or caused the incident), and and entry called supplementary, for information that does not fit neatly in any of the previous categories, like problem areas (possible ones being human performance, environmental factors, company policies). There can be more than one entry for each of aircraft, person, and component type of entries, one for each such entity included in this part of the record. All others show exclusively a single entry in all records we have encountered.

The second part of the record consists of plain running text and is a description of the incident in English prose.

The origin of each database record is a form which consists of a first part, with checkboxes and blanks to fill in.[1] This provides some of the structured information in the final database records. The second part of the form is simply a blank page where the reporter of the incident is supposed to describe it. This, with some modifications introduced by ASRS's experts (mostly to strip out any identifying details) is what constitutes the unstructured part of the report.

Some of the information in the structured part clearly comes from the structured part of the submission forms. However, there are, in most reports, a number of entries that clearly come from somewhere else. The relation between these and the text is explained below. This information comes either from the analysis of the ASRS experts, or from subsequent contact with the reporter of the incident.

The report text contains a number peculiarities specific to reports of this type. These

[1] Copies of the forms used to report incidents to ASRS can be downloaded from http://asrs.arc.nasa.gov/forms_nf.htm.

include:

- field-specific lingo: words like *encroach*, *taxi*, *thrust*, are specific to the field of aviation or take special meanings within it.
- abbreviations: these documents are riddled with abbreviations. Some examples are *MGMT* for management, *ACFT* for aircraft, *INFLT* for inflight, *RWY* for runway.

4.3 Use of the Database Information

Analysis of a few reports selected at random from two collections available online from the ASRS website reveals some relevant issues about the relation between the structured and unstructured part of each record.

The proportion in the amount of information between the two parts is largely unpredictable. Some lengthy reports have short structured headers, some short ones have long headers.

For example, report ACN533918 has an unstructured part only five lines long. It explains, very briefly, an incident derived from loss of cabin pressure. It states that an emergency is declared, following which a landing clearance is obtained immediately, and landing proceeds uneventfully and according to procedure. The only other thing mentioned is a notification that another plane is nearby, with which visual contact is never established.

The structured part of this report, however, is over average in length. A component entry for the pressurization system is only implied by the report, which speaks about loss of pressure, never about the pressurization system. There are four person entries, however only two are mentioned: the reporter of the incident, only in first person, and the air traffic controller, in "LATER NOTIFIED BY *CTR* THAT OUR ACFT WAS WITHIN 20 SECONDS OF AN ACR AIRLINER (...)". CTR, meaning control, is the air traffic controller. The other two people are only implied by the use of plural rather than singular first person throughout the report.

The fact that the plane is diverted to another airport is also mentioned in the structured part and not in the text. There is, however, a synopsis of the report below the report text which does mention the fact, but the original writeup by the reporter does not contain it. The landing is mentioned in the structured part and also only implied in the text:

INITIATED EMER DSCNT INCLUDING TURN PER AIRCRAFT STANDARD PROCS.

Other reports, on the other hand, have rather short structured headings in comparison with the text in the unstructured part. Report ACN532693, for example, has a rather lengthy header as well, but an even longer text. This report describes a landing gear problem derived from non-adherence to company policies and published procedures. The writer of this report is clearly distressed by the incident, and goes into detail of everything that occurred leading to and during the incident. This includes an excerpt of a conversation that he had with the captain (he was first officer) about how they both had little experience on that type of plane, his suggestion to report this to scheduling, and the captain's refusal. There is a Person entry for the captain, but it says nothing about his experience. The Supplementary entry states that the company and flight crew human performance are "Problem Areas", without further details. The report includes an interesting account of how the company fails to keep basic safety standards when the captain dismisses the first officer's idea to call scheduling and tell them that neither of them is experienced enough on this type of plane.

There are certain entries in the header that come from the structured part of the form and are rarely mentioned in the text unless they directly concern the incident. Examples of such are plane make and model, ratings and experience of the reporter, plane mission (passengers, cargo, ...), operator and flight plan.

Some of the information almost always occurs in the structured part but is repeated in the text more often. Examples are location information (airport, city; airborne, on runway, in garage, etc). Weather conditions will only be mentioned if relevant, specially if they are part of the problem. Good weather conditions are always in the headers but rarely ever in texts.

Information first introduced in the text (i.e., not in the header) mostly comes in the form of a relation of what happened, usually rich in chronological, geometrical or human-interaction information (or a combination of these). The headers can only give a gross classification of the type of incident due to their finite and highly structured nature.

For example, report ACN538691 relates an airborne conflict (two planes coming unsafely close to each other, yet not close enough to classify it as a Near Mid-Air Collision (NMAC)). While the header does provide many facts about the incident, including that there was an

airborne critical conflict, that evasive action was taken, that the assigned course was eventually retaken, as well as all the information that all records have regarding the people involved, the weather conditions, the time and place, and information on the aircraft involved, there is a wealth of information in the text about the chronological succession of facts, with details of the positions of each plane and their position relative to the runway at the relevant moments.

4.4 Our Corpus

The corpus used for this research contains the unstructured parts of 5,831 ASRS reports. The sentences in this report are parsed by Cass according to the e8.reg grammar (see Section 3.5.4). Here is an example sentence from the corpus:

```
<phrase tag="#phrase">
  <phrase tag="#c">
    <phrase tag="#c0">
      <phrase tag="#nx">
        <word base="wind" tag="/nn">WIND</word>
      </phrase>
      <phrase tag="#vx">
        <word base="be" tag="/vbd">WAS</word>
      </phrase>
    </phrase>
    <phrase tag="#nx">
      <word base="090" tag="/nn/car">090</word>
    </phrase>
  </phrase>
  <word base="/" tag="/sym">/</word>
  <phrase tag="#nx">
    <word base="8" tag="/jj/car">8</word>
    <word base="kts" tag="/nn">KTS</word>
  </phrase>
  <word base="." tag="/&amp;quot;.&amp;quot;">.</word>
</phrase>
```

The markup is quite straightforward: <phrase tag=*tag*> indicates the opening bracket of a constituent of type *tag*. The closing bracket of a constituent is indicated by </phrase>. Individual words are enclosed between <word base=*base* tag=*tag*> and </word>. The attributes *base* and *tag* refer to the word's base form, which we do not use in this project, and the word's part-of-speech tag, see Section 3.2, which is used by `tagfixes` and Cass. Indentation shows clearly the hierarchical containment of constituents. When we mention 'the corpus' in subsequent sections of this document, we will be referring ourselves to this particular file.

Chapter 5

Room for Improvement in the ASRS Database Corpus

By means of direct corpus inspection, a few weaknesses of the `e8.reg` grammar were found and will be discussed in this section.

Some of the problems in parsing arise from errors in tagging. The POS tagger used, the Church Tagger [10] is not 100% accurate. Being a process previous to parsing and handed down to us as is, there is nothing we can really do about these errors in the context of the present project. The only thing that could be done to improve the accuracy measures in the modified grammar would be to allow for these errors in the corpus data in the golden standard by coding the errors into the standard.

This would, however, render the measure somewhat undefined. What are we really measuring if our golden standard, supposed to reflect our ultimate notion of correctness, has errors deliberately coded into it? As we will see in the next section, when we discuss how the golden standard was put together, we do make some concessions of this kind, but only in cases where we can strongly argue that this constitutes a fair gauge for a partial parser, whose output structures are inherently limited (see Section 3.4).

The following sections discuss, one by one, the different misparsings we found in the corpus and decided to address by means of grammar modification.

5.1 Treatment of the Forward Slash

The first problem we will discuss is that of the forward slash ("/"). This particular character occurs very frequently in these reports, as it occurs in date expressions, a vital piece of information in incident reports.

The sentence that we chose from the corpus to illustrate this is:

4/88, I DEPARTED BOEING FIELD

The phrase "4/88" stands for April, 1998. The day was likely removed prior to processing in accordance with the ASRS policy of removing exact dates and other pieces of information that could be used to identify the people involved. The parsing provided by the corpus being:

```
[#nx 4]
/
[#nx 88]
,
[#nx I]
[#nx [DEPARTED/vbn]
     [BOEING/nn]
     [FIELD/nn]]
.
```

The phrase '*departed Boeing field*' is wrongly parsed because '*departed*' is mistagged as a past participle, but that is immaterial for the purposes of the present discussion. More interestingly, it can be observed that the slash was left unattached at the top level of the parse tree. The ideal parsing for this type of expression considers the whole expression "4/88" as a single chunk. Since the original grammar comes with a category specifically for date expressions, it would be desirable to use it, yielding a parse like this:

```
[#date 4 / 88]
```

Another kind of date expression that was found in the corpus looks like this:

```
[#pp FOR [#nx 3]]
/
[#nx SAT]
/
[#nx 88]
```

Clearly this means "Saturday, March 1988". This phrase occurs completely broken up into its individual constituents, only its first one being attached to the prepositional phrase that they all should be contained in, grouped together as a single constituent.

After the fixes that we perform in the grammar, this is what this phrase looks like:

```
[#pp FOR
  [#date 3 / SAT / 88]]
```

The problem with this construction is satisfactorily addressed in this project, as detailed in Section 6.1. However, it cannot be claimed that all forward-slash-related errors were addressed. Although dates are likely to be a prevalent source of error in this corpus, because they do seem to occur very often in these reports, this symbol appears in a number of different situations that we did not attempt to correct due to restrictions of time and space.

One such situation listed here for example's sake is seen in this phrase from the corpus, which ended up in the golden standard:

> THE CAPTAIN AGREED, EVEN THOUGH I STATED THAT I HAD NEVER TAKEN AN WDB WITH 2 L OR 2 R INOP W/O STICKING BOTH THE INBOARD AND OUTBOARD TANKS AS PER MY UNDERSTANDING AND TRNING IN THE MEL.

The problem here is with the word "W/O". It is a shorthand for "without", but because it has the slash in the middle of it, the tagger splits it into three words, confusing the parser.

This could be addressed by having "W/O" as a lexical entry. This solution, however, would involve substantial modification of the grammar and its complicated implementation seemed overkill for a problem which in reality is not prevalent in the corpus.

The pertinent modification would have seen a new grammatical symbol `#prep` added to the grammar. The production defining it would have been placed at the bottommost level, the `:nx` level, and it would have looked like this:

```
prep -> w slash o
```

The symbols `w` and `o` would be defined by means of respective tagfixes:

```
w       w       nn
o       o       nn
```

This would have changed the tags of the occurrences of `W` and `O` around the slash, which the tagger classified as nouns, to `w` and `o`. This would have caused our production to 'wrap' the `W/O` in a constituent like this:

```
[#prep W / O]
```

Now we would have a new grammatical category, `#prep`, which is equivalent to the terminal `/in`. Unfortunately there is no other way to equate these in the grammar than by brute force: we would have to:

1. define, at all levels of the grammar in which at least one production contains `in` as a terminal, an alias `PREP = prep | in`,

2. look for all occurrences of `in` in right-hand sides of productions in the grammar and substitute them by `PREP`.

Of course this leaves us with the problem that if 'W' or 'O' were to occur somewhere in the corpus as legitimate nouns, we would have to implement a similar fix to indicate to the grammar that a noun could now come labeled `/nn`, `/w` or `/o`.

This solution clearly is too cumbersome for the benefit it provides, specially if one takes into account how much it complicates the grammar maintenance task to have *ad hoc* symbols defined like that for circumstantial reasons. In general, one should try to keep one's symbols as linguistically relevant as possible.

Problems like this will not be addressed in this project. The golden standard features the correct parse with the whole expression taken as a single word and tagged as a preposition

(/in), but our revised grammar will not generate that parse and no performance gain will be reflected in this particular utterance.

5.2 More on Punctuation: the Single Quote

The single quote (') is frequently used in these reports to signify feet as a measure of altitude. Let us look at the following sentence, which also ended up in the golden standard:

> AFTER RECEIVING MY TRANSPONDER CODE, I LEVELLED OFF AT 1500'.

And the analysis in the corpus is:

```
AFTER
[#vgp RECEIVING MY TRANSPONDER CODE]
,
[#nx I]
[#nx LEVELLED]
OFF
[#pp AT [#nx 1500]]
,

.
```

This sentence is quite unsatisfactorily parsed due to an unfortunate convergence of mistaggings. Notice, for example, how '*levelled*' occurs isolated in a noun phrase because it was labelled as a noun.

What concerns us here, however, is that the single quote appears completely unattached, when ideally it should be two levels deeper inside the prepositional phrase '*AT 1500*', in particular grouped together with the cardinal '*1500*' as a measure phrase, an existing category in the original grammar.

The desired parse for the phrase "*AT 1500'* " would look something like this:

```
[#pp AT
     [#mx 1500 ']]
```

This problem is successfully addressed in this project. Section 6.2 provides a detailed explanation of the solution devised.

5.3 Further Punctuation Problems

A problem similar to the one found with the two previous symbols can be observed with quotation marks (") and brackets ("(" and ")"). From the point of view of partial parsing, quotations marks, brackets, and any other pair of symbols that match at the ends of a construction they enclose can be considered homologous in which they will be given similar treatment. Therefore, this section will only discuss the case for quotation marks, and we will claim that any conclusions obtained will also be valid for the brackets case.

Let us take a look at the following sentence:

> DO YOU HAVE TO STATE IN LNDG CLRNC "DO NOT TURN OFF ON RWY 28"?.

The provided parse is:

```
[#vp DO]
[#c [#c0 YOU HAVE]
    [#pp TO STATE]
    [#pp IN HIS LNDG CLRNC]]
"
[#vp [#vx DO NOT TURN]
     OFF
     [#pp ON [#nx RWY 28]]]
"
?
.
```

The reported phrase "do not turn off on rwy 28" should appear at the same level as the `#c0` and the two `#pp` phrases, since it is an argument to the verb head "STATE".

At first glance this problem seemed like an easy target for correction via grammar modification. Section 6.3, however, shows why this is not the case. This problem was left unsolved in the present project.

Another, unrelated problem, concerns the use of quotation marks as a measure unit meaning "inches". This problem was also left unsolved, but it might be easily addressed in a way similar to the single quote as a symbol for "feet". Interactions with occurrences of

quotation marks as reported speech delimiters are possible. Due to limitations of time and space, they were not looked into.

5.4 The Problem with 'away'

The following construction is misparsed in the corpus:

```
[#vp [[#vx WAS]
      MANY/rb]]
[#nx MILES]
[AWAY]
```

This analysis leaves much to be desired. There are two different sources of syntactic error in this phrase. One is the problem with post-head adverbial modifiers like 'away' here, which we discuss in this section. Another is the handling of 'many', which is discussed below in Section 5.5.

The adverb 'away' is left out of the analysis altogether, leaving it completely unattached, a partial parser's last resort and an indication that this construction is not allowed for at all in this grammar. Ideally, this phrase should be parsed as follows:

```
[#vp WAS
     [#ax [#mx MANY MILES]
          AWAY]]
```

That is, the phrase as a whole is still a verb phrase, but 'many miles away' constitutes an adverbial phrase, in which 'many' plays the role of a quantifier for 'miles', and 'away' is attached at the end of the adverbial phrase as a post-head adverbial modifier. The phrase 'many miles' is to be considered a measure phrase, a category found in the original grammar.

A solution that provides an analysis similar to the one shown above is discussed in Section 6.4.

5.5 The Problem with 'many'

As mentioned above, the parse

```
[#vp [[#vx WAS] MANY]
     [#nx MILES]]
[AWAY]
```

is highly unsatisfactory. Besides the problem with 'away' discussed above, another problem is that 'many' is being treated incorrectly. One problem is that 'many' is incorrectly tagged as an adverb. This is specially odd given that *many* can never really act as an adverb. This mistagging forces Cass o take it as an adverbial modifier to the verb *was*. The treatment that the grammar gives to *many* is equally hard to account for. The implemented solution, discussed in Section 6.5, takes advantage of the mistagging of *many*, which is observed in 100% of its occurrences in the corpus, to solve the problem independently from the treatment of correctly tagged occurrences of the word.

5.6 The Problem with 'when'

The following sentence was found wrongly parsed in the corpus:

```
[[#pp WHEN/in I]
 [#vp LANDED]
 ,
 [#pp TO MY SURPRISE]
 ,
 [#c [#c0 THE POLICE WERE WAITING]
     [#pp FOR ME]]]
 .
```

Rather than a prepositional phrase followed by a dangling, unattached verb phrase, a more desirable parse for this sentence would treat the phrase *'when I landed'* as a single

subordinate clause, like this:

```
[[#subc
   [#subc0 WHEN I LANDED]]
 ,
 [#pp TO MY SURPRISE]
 ,
 [#c [#c0 THE POLICE WERE WAITING]
     [#pp FOR ME]]]
```

Section 6.6 shows how this problem was addressed.

Chapter 6

Addressing the Problems

In this chapter we will discuss the grammar to be edited in detail and how we will modify it so that the performance, as measured by the Parseval metric we implemented and discussed in Section 7, will be boosted.

6.1 Dealing with the Forward Slash Problem

The problem with the forward slash ('/'), discussed in Section 5.1, is dealt with only for the case in which it appears as part of a date expression or surrounded by numbers, leaving other problems with the symbol unsolved.

The fix that was implemented for the date case involves several additions to the tagfixes file and to the grammar. The tagfixes file was enhanced with a line that reads

```
slash   /       sym
```

This defines a new tag for the forward slash when it appears tagged "`/sym`". This is the tag with which it occurs in the corpus every time.

The other lines we added to the tagfixes file define a new tag, `/wday`, for expressions like

"3/sat/88" shown in Section 5.1. They look like this:

```
wday    sun             nnp
wday    sunday          *
wday    mon             *
wday    monday          *
wday    tue             *
wday    tuesday         *
wday    wed             *
wday    wednesday       *
wday    thu             *
wday    thursday        *
wday    fri             *
wday    friday          *
wday    sat             *
wday    saturday        *
```

The new tags `slash` and `/wday` are used in two productions in the grammar. In the original version, dates are handled by a production that reads:

```
date -> month cd (cma cd cma?)?
```

This deals effectively with date expressions like "April 3" or "April 3, 1945". The tag `month` comes from the tagfixes file:

```
month   jan.            nnp
month   january         nnp
month   feb.            nnp
month   february        nnp
            .
            .
            .
month   dec.            nnp
month   december        nnp
```

Other forms, which are prevalent in our corpus, are not handled well. The two forms of date expressions that we find in our golden standard are

- a succession of two or three numbers separated by slashes (as in "6/14" for June 14, 4/88 for April 1988, or "6/14/76" for June 14, 1976),
- a weekday name succeeded by one or two numbers, all separated by slashes (as in "TUE/6/88" for Tuesday, June 1988).

The former case is impossible to distinguish from other expressions involving numbers, such as number ranges or wind expressions. Hence, it is dealt with outside of the production for dates. The aim is set at keeping numbers involved in the expression clustered together as a unit, even if the constituent label obtained is not completely specific. The solution is discussed below in Section 6.7. This moved us to modify the grammar to handle these cases properly. The latter case listed above was handled successfully by adding one case to the production for dates:

As for the former case, we added the following case to the production for dates in the grammar:

```
date -> month cd (cma cd cma?)?
      | cd slash wday (slash cd)
      ;
```

The added line deals with expressions like the one in the following examples. In the pre-edition corpus, we find:

```
[#pp FOR
  [#nx 3]
]
/
[#nx SAT]
/
[#nx 88]
```

In the post-edit corpus, this becomes:

```
[#pp FOR [#date 3 / SAT / 88]]
```

Expressions like "4/88" for "April 1988" are dealt with separately. Their treatment is discussed below in Section 6.7

6.2 Dealing with the Single Quote Problem

The problem with the single quote ('), discussed in Section 5.2, is dealt with by a simple addition to the tagfixes file.

The original grammar contains a production that satisfactorily deals with expressions involving measure units:

```
## Measure Phrase Chunk
mx -> (cd|cdx) h=(units|tunits)
   |  (cdql|cdqlx)? dt-a h=(unit|tunit)
   ;
```

This means that all that needs to be done for expressions like the ones exemplified in Section 5.2 to be parsed as desired is to include the single quote in the `units` category.

To accomplish this, a single line is added to the tagfixes file. The portion of the file that deals with the relabelling of measure units consists of lines like these:

```
units   meters          nns
units   millimeters     nns
units   yards           nns
units   feet            nns
```

Next to all these, a line will be added that reads:

```
units   '               '
```

By virtue of the new tagfix and the existing grammar production, the following parse:

```
[#subc
  [#subc0 THAT
    [#nx I]
    [#nx WOULD BE]]
  [#nx DSNDING/nn]
  [#pp TO
    [#ng [#nx AN ALT]
         OF
         [#mx 500]]]]
'
```

found in the pre-edition corpus was turned into

```
[#subc
  [#subc0 THAT
    [#nx I]
    [#nx WOULD BE]]
  [#nx DSNDING/nn]
  [#pp TO
    [#ng [#nx AN ALT]
         OF
         [#mx 500 ']]]]
```

in the post-edition corpus.

6.3 The Quotation Marks: a Problem too Complex to Solve Here

The problem with the double quotation marks (‘"’), discussed in Section 5.3, appears to be simple at first glance. However, as it turns out, addressing this problem with this kind of parser is way more problematic than it seemed in the first place. The difficulty lies in the fact that this parsing technique builds structures bottom-up rather than top-down. So, it is not possible to isolate an expression between quotes, and then break it up into its constituents: rather, one should allow for the possibility of finding the expression being built surrounded by quotes at any point up the levels of the grammar.

Dealing with quoted text in general is far from being a trivial problem, as illustrated by [12]. And this paper discusses dealing with quoted text with Tree Adjoining Grammars, whose expressive power is ranked above that of Context Free Grammars in the Chomsky hierarchy. Regular Grammars, used by Cass, rank below CFGs. The use of powerful TAGs makes it easy to bestow hierarchical structure on constructions with matching pairs of quotation marks. Achieving this with cascaded regular grammars is, unfortunately, not that simple. In fact, the language of balanced parentheses, $\{(^n)^n\}^*$, is not even a regular language, so the best we can hope for is recognize only formations of a number of nested brackets fixed at grammar creation time. But even dealing with a single pair of matching brackets or quotation marks proves extremely cumbersome, error-prone and unsatisfactory in terms of achieved expressive power, as the following discussion demonstrates.

To illustrate some of the complications of dealing with this we will take a look at some potential solutions to the problem and see, for each of them, how they are either unsatisfactory or infeasible.

The first solution we will consider is the simplest approach: we could define, at the bottom level of the grammar (`:nx` in `e8.reg`; we could also add a new level right underneath it), a symbol *junk* meant to deal with quoted and bracketed strings:

```
:nx
   ANYTHING = nn | nnp | nns | vb | ... ##the WHOLE tagset goes here

   junk -> quote ANYTHING* quote
```

This is similar to the treatment given to quoted text in [16]. The *junk* terminal is not to occur anywhere else in the grammar, so the quoted text will be in a single flat cluster, with the quotation marks in it. That is the only advantage of this method, which is largely unsatisfactory because structural information inside the quoted text is lost, something the grammar as it stands right now does quite well. Hence, there is really no gain from implementing this solution for this project. Yet another problem with this approach is that the implementation of the alias ANYTHING is brute-forced, inelegant, and error-prone.

Another possibility that was investigated involved looking through the corpus for typical sequences of grammatical constructs that would appear between brackets or quotation marks. The idea was also taken from [16], where it is stated that "*JUNK* was allowed to

occur wherever a sentence modifier, post-nominal modifier, or a pre-nominal adjective could occur". Since shallow parsing does not concern itself with this dependency information, the reasoning followed, the closest thing we could achieve here is to allow for the constituent types that normally make up phrases with such dependency labels to be surrounded by quotes or brackets.

As it turns out, noun phrases, adjectival phrases, and adverbial phrases are the constituent types most frequently filling those dependential roles in sentences. An inspection of the corpus was in turn, to verify, at least preliminarily, that the expressions found quoted or bracketed in the text actually corresponded to the counterparts of these constituent types.

The results were very disappointing. The following sentence is a good example of the kinds of expressions that can be found between quotation marks in the corpus:

> A SIMPLE "TAXI DOWN RWY 13, 17 ITXN VIA HOTEL" WOULD ALERT THE PILOT ...

The sequence of grammar symbols assigned by the parser, even assuming proper tagging ("TAXI" is labelled as a noun here, for example, when it should be a verb) would be something like `#vp #pp cma #nx #pp`. This sequence does not seem to match the right side of any existing grammar production, or have any kind of linguistic significance.

Another interesting point to make towards the infeasibility of this solution: even for a toy sentence like "She said 'I love you'" this approach would involve adding a whole new level to the grammar. This brings about a number of potential problems derived from the fact that it is a major, error-prone modification.

The idea would be to add a whole new level in between the clause (top) level and the one right underneath it, the `:c0` level. This new level would have a production with a new symbol, called `#c1`, on the left side, representing a quoted clause. Its right hand side would be just like the right hand side of the `#c` (clause) production one level above it, but surrounded by quotes.

```
c1 -> quote h=c0 o=NOM? ADV* quote
```

On meeting a perfectly formed clause surrounded by quotes, the parser will label it c1 and structure it as a single chunk. If a clause is there, but not surrounded by quotes, it will be ignored at this level.

The next level will now look something like this:

```
NOM = ng | nmess | ... | c1

c -> h=c0 o=NOM? ADV*
...and so on...
```

So now, a perfectly formed clause that might have gone unmatched in the previous level will be labelled and structured as a `#c` just like before. But a quoted clause, which will have been labeled `#c1` in the previous level, will now be liable of the same treatment as a noun phrase. The idea is for the quote to be able to appear as, for example, the direct object of a sentence. Hence the above example will now be parsed like this:

```
[#c [#c0 she  said]
    [#c1 " [#c0 i love you ] " ]]
```

In this parse, the quoted text appears as the object of the verb "say", a satisfactory analysis.

However, this fix is only good for a matching pair of quotation marks appearing around a perfectly formed clause. Hence it is likely that the potential drawbacks of this solution, like potential interference with other aspects of the grammar that work well without the modification, might outweigh its benefits. In particular the golden standard has not a single sentence with these characteristics. Hence, the net gain from implementing this solution here would be zero in the best case for our project.

Yet another problem is that quotation marks also appear on their own, with no matching counterpart, as a unit measure meaning "inches".

6.4 Solving the Problem with 'away'

To correct this flaw, we first need to come up with a desired parse in terms of the categories and symbols used in `e8.reg`. It is clear that the construction 'many miles away' should be treated like an adverbial phrase by the parser.

Several levels of the `e8.reg` grammar have a macro that stands for adverbial phrases, dubbed ADV. Its definition varies across levels of the grammar. At the top three levels, `c1`,

c0, and rc, the definition is

```
ADV = rb | cdql | then | well | rbr | more | rbs | ql | such
    | pp | pp-comp | rx | ax | in | tadvx | today | date;
```

The only other level in which and ADV macro is defined, five levels below rc, is the :nx level, and its definition there,

```
ADVHD = rb | cdql | then | well;
ADV = ADVHD | rbr | more | rbs | ql;
```

is just a subset of the definitions in levels above it. Hence, we will inspect the categories in the definitions at the top three levels and claim that any relevant conclusions will also apply to the definition at the :nx level below.

The categories listed under ADV in the top three levels can all be easily concluded to be able to fill the role of an adverbial construction in a sentence. Of particular interest are the categories rx and tadvx, at the :mx and :nx levels respectively, which are meant to parse adverbial phrases, as is apparent from their definitions:

```
:mx
   tadvx -> ( cdx? h=(nns|units|tunits)
            | (cdql|cdqlx)? dt-a h=(nn|unit|tunit)
            )
            ago
            ;

:nx
    rx -> ADV+ ADV
        |  by then
        |  MX ago
        ;
```

And the definition for the MX macro occurring in the rx production is

```
MX = mx | units | tunits;
```

where mx, at the :mx level, is

```
mx -> (cd|cdx) h=(units|tunits)
   |  (cdql|cdqlx)? dt-a h=(unit|tunit)
   ;
```

In conclusion, the only differences between these two productions, in terms of the kinds of constructions that they will parse and the structures that they will bestow upon them, are the contexts in which they will be used (the two productions occur in different places around the grammar), and the fact that the tadvx production allows for any nouns, not only units of measure, to be used. This allows for phrases like "We should have got off the train many stations ago". But other than that, they both yield very similar parses and will be treated alike for the purposes of the present discussion.

The definition for rx is at the :nx level, down below in the grammar where the ADV macro takes its shorter from, see above. The one for tadvx is at the :mx level, the bottom-most level of the grammar. Both these productions yield parses whose structures remarkably resemble that of the construction we are trying to parse.

There is another fact that prevents this grammar from giving a correct parse for this construction. The phrase "3 miles" will be satisfactorily parsed as a unit of measure by the mx production which occurs in MX above. The phrase "many miles" however, will not, but this is due to a separate problem with the treatment of "many", which is discussed below.

The symbol 'ago' actually stands for a number of different words in the tagfixes file e8.fx, as show here:

```
ago     ago     *
ago     later   *
ago     earlier *
ago     sooner  *
```

So the adverbs ago, later, earlier, and sooner are all treated alike. One objection that might be raised against this treatment is that, unlike with the rest of the adverbs in the category, 'ago' cannot be used with prepositional phrases or in comparative constructions with 'than':

That happened 3 hours ago.

*That happened 3 hours ago from now.

*That happened 3 hours ago than this.

This is a very valid objection, but of no significance for a partial parser, which tries to confer structure to every chunk it can find in the sentence. Hence the incorrect sentences will be given a parse, which will treat the 'from now' or the 'than this' as an unattached prepositional phrase, the same treatment that would be given to a correct phrase.

These three adverbs can be treated similarly in the context of the present project, however. This is due to their ability to appear in constructions with post-head adverbial modification, such as:

The boys stood 3 feet apart.

The boys left 3 hours ago.

The plane was 3 miles away.

In the light of the above described evidence, a fix for this problem would be as simple as mapping the tag for the adverb 'away' into 'ago' in the tagfixes file. The modification would see this line included in the file:

```
ago      away   *
```

A similar line was introduced for 'apart'.

In the Section 6.5 we show an example from the corpus, containing the expression 'many miles away' that motivated the discussion in the present section, and how the new grammar does indeed successfully handle the construction.

6.5 Solving the Problem with 'many'

Section 5.5 discusses the problem with the treatment of the adjective *many*. In this section we propose a solution to that problem.

In our example phrase 'many miles away', 'many' is being mistagged as an adverb. This results in e8.reg treating it like an adverbial modifier of the verb 'be' in 'was many'. The noun 'miles' is left out of this bracketing, thus not reflecting that 'many' is actually modifying 'miles'.

While `e8.reg` and `e8.fx` do include productions and tagfixes to deal with 'many', they would not deal with the phrase 'many miles away' appropriately even if 'many' were properly tagged. To see this, let us investigate the contents of the original grammar and tagfixes file once more.

The tagfixes file contains the following lines:

```
dtp     that    dt
dtp     this    dt
dtp     these   dt
dtp     those   dt
dtp     few     dt
dtp     several dt
dtp     much    dt
dtp     many    dt
dtp     many    jj
dtp     last    jj
dtp     next    jj
```

It is arguable whether 'many' should be tagged as a determiner or as an adjective in our example phrase. But it does not really matter, because both taggings of the word are tagfixed to `/dtp` here by `e8.fx`.

The places in the grammar where `/dtp` occurs are three:

```
:mx

    timex -> dtp h=tunit;
```

at level :mx. Then at level :nx we have

```
:nx
    DET = dt | dtp | prps | (cdql | cdqlx)? (dt-a | dt-q | dtp-q);

    nx -> such? DET? NUM? (ADJ | PTC)* (ADJ | N)* h=COMMON cd?
       |  DET? NUM? (ADJ | PTC)* h=PROPER
       |  DET h=(jjr | jjs | such)
       |  cdql? h=dtp-q
       |  h=( prp | cd | dtp | qq | ex                # prp=Personal Pronoun
            | name | person | doll | ci-st | rbr | rbs
            )
       ;
```

and finally, at level :nmess we find

```
:nmess

    DET = dt | dtp | prps cdql? (dt-q | dtp-q);
    NOM = nx | mx | cdx | place | person | name | ci-st | doll | date;

    nmess -> DET ax* h=NOM?;
```

None of these productions matches the pattern '/dtp /nn /rb' and creates the structure[1]

```
[#rx [#mx MANY MILES]
     AWAY
]
```

So the grammar as it stands would not parse this particular phrase correctly even with proper POS tagging. It is clear that the handling of determiner/pronouns (hence the tag dtp) by this grammar precludes correct analysis of this particular phrase. However, a

[1]Notice that the labels chosen in this intended parse could be replaced by others and we would still consider the analysis correct: for example, we label 'MANY MILES' as an #mx because this is a category that this grammar defines for expressions involving units of measure, but #nx, the label for a noun phrase, would still be appropriate, if less specific.

circumstantial fact, related to the corpus data, may allow us to come up with a "quick fix" that works around this problem, with the added advantage that it avoids a reclassification, a drastic measure that might make the parser give incorrect parses for sentences which the grammar as it stands analyzes correctly.

The observation is that ALL occurrences of 'many' in the corpus are tagged as adverbs rather than adjectives. We could thus take advantage of the fact that neither of the tagfixes for 'many',

```
dtp     many    dt
dtp     many    jj
```

which convert its tag to `dtp`, expect an adverb. Thus the addition of the line

```
cd      many    rb
```

to e8.fx would consistently convert the tag to a cardinal, producing a correct analysis in every case.

This also has the added advantage that it does not modify the existing handling of 'many', for which, as mentioned above, there might be a rationale that is simply unknown to us. This consideration is specially important if the fact is weighed in that Cass grammars, as conceived by Abney, are to be looked at more as a "programming language" for grammars than as a traditional grammar with linguistically motivated productions. Hence, linguistically correct classification of utterances is of secondary importance to conferment of useful structure, which Abney's original grammar e8.reg likely does satisfactorily for a number of frequently occurring constructions that do not come up in the present discussion.

The following example taken from the pre-edition corpus shows a construction involving both 'many' and 'away':

```
[#vp [#vx WAS]
     MANY]
[#nx MILES]
AWAY
```

In the post-edition corpus, this becomes:

```
[#vp [#vx WAS]
     [#rx [#mx MANY MILES]
          AWAY]]
```

which is exactly the desired parse.

6.6 Solving the Problem with 'when' as a Subordinating Conjunction

The problem here is that 'when' is being treated as a preposition. This problem is due to the fact that the Penn Treebank tagset does not distinguish subordinating conjunctions from prepositions. This is because in the Penn Treebank the information to make that distinction is coded into the parse trees. Both parts of speech are labelled `/in`; however, prepositions precede a noun phrase or a prepositional phrase, conjunctions precede clauses. This forces the grammar in treat the phrase *'when I'* as if it had a structure similar to that of the phrase *'with me'*.

The grammar handles this problem by means of a collection of simple tagsets that pick particular words that come tagged `/in` and change the tag to `/comp`. The tag `/comp` is then used in the grammar as a start token of the productions that deal with subordinate clauses. These are the lines in the tagfixes file that perform said tag change:

```
comp    that       in
comp    if         in
comp    while      in
comp    whether    *
comp    though     *
comp    although   *
comp    once       in

p-comp  before     in
p-comp  after      in
p-comp  until      in
p-comp  since      in
```

There are two kinds of subordinating conjunctions, `comp` and `p-comp`. The difference between them is that subordinate clauses starting with `p-comp` may have no subject, as in *'after leaving high school'*, while ones starting with a `comp` must have a subject in this grammar.

This shows that most subordinating conjunctions are being expected with the tag `/in`, but their tags are being changed to `/comp` prior to parsing. The original grammar contains productions that satisfactorily deal with subordinate conjunctions:

```
:c0
   subc0 -> in? (s=pp-comp | (comp | because | wrb) s=NOM) SUBJ-TAIL h=vx;

:c1
   subc -> h=subc0 o=NOM? ADV*;
```

as we can see, a subordinate clause *may* start with a preposition (as in *'the man with whom we talked last night wears a hat'*), which can be followed by one of *before*, *after*, *until* or *since* (under the tagfix `p-comp`), or by *because*, or by a wh-adverb, namely one of *what*, *who* and *whom* when used as complementizers. This can be followed by `NOM`, which is a local alias the grammar uses to engulf noun phrases in all their possible forms. The remainder of the productions is immaterial for the purposes of the present discussion.

The idea is that *when* could have a line in the tagfixes file that transforms it into a `comp`. This is arguably linguistically sound, since *if*, *while* and *when* all classify as subordinating conjunctions when they precede a clause. But most importantly, it is a modification that will achieve the desired effect of treating phrases like the one in the example as a single subordinate clause rather than a prepositional phrase followed by an unrelated verb phrase.

One objection that could be raised against this change is that *'when'* can occur with functions other than a subordinating conjunction. The other senses of the word, however, are very rare, and chances that it might occur with a function other than a conjunction are comparable to the chances of *if* occurring as a noun (meaning a condition or stipulation). Finite-state grammars only seek to find useful structure in as many cases as possible, rather than find a perfect parse every time. With all this in mind, when clearly should have a line in the tagfixes file to give it the same treatment as any other subordinating conjunction.

This fix is indeed successful in recognizing subordinate clauses that were being mishandled so far. For an example from the corpus, take this parsed phrase from the pre-edition

corpus:

```
[#pp WHEN [#nx I]]
[#c-inv [#vx REACHED]
        [#nx THE WATER]]
```

The above described addition to the tagfixes file changes the parse of this very same phrase to

```
[#subc
  [#subc0 WHEN
    [#nx I]
    [#vx REACHED]]
  [#nx THE WATER]]
```

which is the analysis we wanted to obtain.

6.7 Untested Fixes

6.7.1 Numeric Ranges, Wind Expressions and Dates Containing Only Numbers

As we saw in Section 6.1, the forward slash is not properly handled by the original grammar. In that section we discussed it in the context of dates. There are, however, a few other contexts in which the symbol occurs which are not parsed satisfactorily by `e8.reg`.

Here is one example:

```
[#phrase
  [#c [#c0 ATC GAVE] US]
  [#ng [#nx A BLOCK ALT]
       OF
       [#nx FL 290]]
  /
  [#nx 310]]
.
```

This parse breaks up what clearly is meant as a number range (290 to 310 feet). A more desirable parse would look like this:

```
[#phrase
  [#c [#c0 ATC GAVE] US]
  [#ng [#nx A BLOCK ALT]
       OF
       [#nx FL 290 / 310]]
]
.
```

This analysis the whole range expression into the same `#nx` constituent.

Another number-range expression which occurs in the corpus consists of two numbers separated by hyphens (-). To allow for proper processing of these hyphens in the grammar, a new tag was added by means of an addition to the tagfixes file. The new line in the tagfixes file reads

```
dash    -     :
```

The tag : is the one defined in the UPenn Treebank Tagset for sentence-internal punctuation, see Table 3.1.

Expressions of numeric ranges consisting of numbers separated by a slash clearly cannot be told apart from dates. Since this distinction cannot be made without some information from the context, we decided to define an alias at the `:mx` level of the grammar that reads

```
CDR = cd slash cd | cd dash cd | cd;
```

which would yield the 'flat' parse shown above, i.e., the numbers and the slash would not be put one level down the parse tree in a constituent of their own.

This way, the desired groupings are kept, even though the name of the constituent they will end up in may not be very specific as to the kind of expression found inside it. The distinction can be carried out by a later processing stage that makes use of semantic and contextual information. So now, a date expression like "4/88" for April 1988, which

currently would be parsed (assuming proper tagging) as

```
[#phrase
  [#nx 4] / [#nx 88] ,
  [#c [#c0
        [#nx I]
        [#vx DEPARTED]]
      [#nx BOEING FIELD]]]
```

will be parsed as

```
[#phrase
  [#nx 4 / 88],
  [#c [#c0
        [#nx I]
        [#vx DEPARTED]]
      [#nx BOEING FIELD]]]
```

Notice that the date expression is labelled `#nx` rather than the more specific `#date`.

This modification addresses the problem satisfactorily, and the degree to which it does so is discussed in Section 7.

Chapter 7

Evaluation and Discussion

The results obtained from running our own implementation of the Parseval metric quite clearly spell an improvement. Table 7.1 shows the overall figures obtained:

	Pre-Edition Corpus	Post-Edition Corpus
Mean Precision	0.713312	0.771012
Mean Recall	0.730579	0.787124
Zero Crossings	8	13
Mean Crossings	3.46	2.54

Table 7.1: Final Evaluation Results

The per-sentence results for the pre-edition corpus are shown in Table 7.2. Table 7.3 shows the same results for the post-edition corpus.

A few facts about this data are worth highlighting here. First, from Table 7.1, it is clear that there was an overall improvement in parser performance from grammar modification. All four figures we have computed moved in a direction of improvement. Average precision and recall rose, mean crossing brackets decreased, and the number of sentences with no crossing brackets with respect to the golden standard went up by three.

A comparison of Tables 7.2 and 7.3 offers a few more interesting facts providing information about the relative performance of the parser on individual sentences. In eleven of the fifty sentences there was a decrease, rather than an increase, in one or more of the figures. In the following paragraphs we will look at one of the sentences whose numbers worsened, look into possible reasons why this happened, and propose solutions.

Sent#	Precision	Recall	Crossings	Sent#	Precision	Recall	Crossings
1	0.400000	0.571429	1	26	0.642857	0.818182	1
2	0.750000	0.800000	1	27	0.714286	0.684211	4
3	0.550000	0.526316	8	28	0.736842	0.750000	5
4	0.692308	0.692308	2	29	0.705882	0.800000	2
5	0.708333	0.695652	3	30	0.705882	0.687500	6
6	0.923077	0.923077	1	31	0.923077	0.928571	0
7	0.761905	0.750000	6	32	0.791667	0.782609	2
8	0.500000	0.470588	9	33	0.666667	0.736842	7
9	0.640000	0.625000	11	34	0.896552	0.896552	3
10	0.625000	0.571429	10	35	0.750000	0.769231	1
11	0.500000	0.521739	13	36	1.000000	1.000000	0
12	0.714286	0.714286	5	37	0.307692	0.400000	3
13	0.692308	0.733333	3	38	0.750000	0.761905	3
14	0.700000	0.777778	1	39	0.680000	0.695652	2
15	0.666667	0.666667	6	40	1.000000	1.000000	0
16	0.615385	0.615385	7	41	0.700000	0.700000	3
17	0.571429	0.700000	3	42	0.789474	0.809524	1
18	0.571429	0.666667	1	43	1.000000	1.000000	0
19	0.600000	0.529412	4	44	0.619048	0.600000	3
20	0.900000	0.923077	0	45	0.500000	0.500000	8
21	0.800000	0.800000	1	46	1.000000	1.000000	0
22	0.875000	0.875000	1	47	0.863636	0.904762	1
23	0.708333	0.681818	7	48	0.714286	0.733333	5
24	0.692308	0.636364	0	49	0.550000	0.578947	3
25	1.000000	1.000000	0	50	0.500000	0.523810	6

Table 7.2: Per-sentence Pre-edition Corpus Results

Sent#	Precision	Recall	Crossings	Sent#	Precision	Recall	Crossings
1	0.750000	0.857143	1	26	1.000000	1.000000	0
2	1.000000	1.000000	0	27	0.850000	0.842105	3
3	0.850000	0.842105	1	28	0.950000	0.950000	0
4	0.846154	0.846154	1	29	0.812500	0.866667	0
5	0.680000	0.652174	5	30	0.705882	0.687500	6
6	1.000000	1.000000	0	31	0.846154	0.857143	0
7	1.000000	1.000000	0	32	0.791667	0.782609	1
8	0.500000	0.470588	11	33	0.769231	0.842105	6
9	0.720000	0.708333	8	34	0.931035	0.931035	1
10	0.857143	0.857143	3	35	0.818182	0.846154	1
11	0.583333	0.565217	9	36	0.666667	0.714286	1
12	0.733333	0.785714	6	37	0.384615	0.400000	7
13	0.666667	0.733333	4	38	0.809524	0.809524	2
14	0.777778	0.777778	0	39	0.680000	0.695652	7
15	0.750000	0.750000	1	40	0.888889	0.894737	1
16	0.769231	0.769231	2	41	0.900000	0.900000	0
17	0.571429	0.700000	3	42	0.882353	0.904762	1
18	0.571429	0.666667	1	43	0.875000	0.888889	1
19	0.764706	0.764706	2	44	0.666667	0.650000	1
20	0.888889	0.923077	0	45	0.714286	0.800000	2
21	0.800000	0.800000	0	46	0.866667	0.875000	1
22	1.000000	1.000000	0	47	0.608696	0.666667	0
23	0.869565	0.863636	1	48	0.928571	0.933333	8
24	0.615385	0.545455	1	49	0.285714	0.315789	11
25	0.875000	0.900000	1	50	0.478261	0.523810	5

Table 7.3: Per-sentence Post-edition Corpus Results

The first sentence we will discuss is sentence 5. Its surface form is

> "AFTER THE PASS BY ALKIA, I DID NOT GO OVER LAND UNTIL I CLBED TO APPROX 1200-1300' PRIOR TO ENTERING THE TFC PATTERN AT BOEING FIELD."

Its parse in the golden standard is

```
[#phrase
  [#pp-comp
    AFTER
    [#nx THE PASS]]
  [#pp BY [#nx ALKIA]]
  ,
  [#c
    [#c0
      [#nx I]
      [#vx DID NOT GO]]
    [#pp OVER LAND]]
  [#subc
    [#subc0 UNTIL
      [#nx I]
      [#vx CLBED]]]
  [#pp TO
    [#mx APPROX 1200-1300 ']]
  PRIOR TO/in
  [#vgp
     [#vgx ENTERING]
     [#nx THE TFC PATTERN]]
  [#pp AT [#nx BOEING FIELD]]]
```

Notice that we chose to take the whole expression "PRIOR TO" as a single preposition. Because it precedes a gerund verb phrase, we decided to leave it dangling at root level, thus reflecting the fact that `e8.reg` does not allow for this kind of structure. Ideally, the preposition would be inside a prepositional phrase chunk with the phrase "ENTERING

THE TFC PATTERN". It was decided to code the golden standard this way so as to set a reasonable standard of correctness for a partial parser. The possibility of a fix that allowed this kind of gerund prepositional phrase was looked into, but abandoned because it was found to clash too much with the rest of the grammar.

The parse that this phrase had in the pre-edition corpus was:

```
[#phrase
  [#pp-comp
    AFTER
    [#nx THE PASS]]
  [#pp BY [#nx ALKIA]]
  ,
  [#c
    [#c0
      [#nx I]
      [#vx DID NOT GO]]
    [#pp OVER LAND]]
  [#pp-comp UNTIL
      [#nx I]]
  [#nx CLBED/nn]
  [#pp TO [#nx APPROX/nn 1200]]
  -
  [#nx 1300]
  ,
  [#ax PRIOR]
  TO
  [#vgp
     [#vgx ENTERING]
     [#nx THE TFC PATTERN]]
  [#pp AT [#nx BOEING FIELD]]]
```

This parse shows a few problems: "CLBED" and "APPROX" are both mistagged as nouns, instead of `/vbd` and `/rb` respectively. The hyphen is not allowed for at all, as number ranges are not considered by this grammar, thus leaving it completely unattached. This isolates the

second number of the range, the "1300". Finally, "PRIOR TO" has its two words tagged separately, so the prepositional phrase is not recognized.

Let us now look at the post-edition version of the sentence:

```
[#phrase
  [#pp-comp
    AFTER
    [#nx THE PASS]]
  [#pp BY
     [#nx ALKIA]]
  ,
  [#c
    [#c0
      [#nx I]
      [#vx DID NOT GO]]
    [#pp OVER LAND]
    [#pp-comp UNTIL I]]
  [#nx CLBED/nn]
  [#pp TO
     [#cdx APPROX 1200]]
  -
  [#ax
    [#mx 1300 ']]
  TO
  [#vgp [#vgx ENTERING]
        [#nx THE TFC PATTERN]
        [#PP AT [#nx BOEING FIELD]]]]
```

Here we can see that what happened with this sentence is unfortunate: because the tagfix that we did for "APPROX" worked so well, the two words "APPROX 1200" was surrounded by brackets and this put it one level down in the tree. This situation was not contemplated when we wrote the solution for number ranges, so now the hyphen is left dangling. The feet expression works fine, but it is not attached to the rest of as we would have wanted.

In conclusion, it was an unforeseen interaction of linguistic constructs that contributed to deteriorate, rather than improve, the score for this sentence.

As we can see, the overall results are satisfactory and future would should focus on fixing the minor flaws that there are. This would take some meticulous tracing of the grammar productions how exactly they match the problematic constructs. This methodical task can be carried out by anyone with expertise in applied formal grammars.

Chapter 8

Conclusion

In this project, we improved an existing grammar for a freely distributed partial parser and geared it to the needs of a particular set of data. This data came from the ASRS database, a program that gathers and analyzes aviation safety reports which consists of some structured data in the form "fill-in-the-blank" slots and checkboxes, and some running English prose.

We first investigated a corpus comprising the unstructured parts of 5,831 reports, looking for constructions which were not given parses that would be useful for subsequent, more semantically informed processing. This task was carried out by manually inspecting the corpus and looking out for things that simply looked misparsed. By analyzing each of the thus picked sentences more rigorously, and in parallel looking the existing grammar for the reasons why the parses found were undesirable, a few of these candidate sentences were taken as examples of linguistic constructs that were not satisfactorily being dealt with by the extant grammar.

Once such linguistic constructs were clearly identified, the possibility was studied of editing the grammar so that structures were conferred to them from which relevant information could be extracted by subsequent processing. For some of these constructs, we found that it was not possible, or at least not feasible, to improve the analysis given by the grammar.

These constructs, concretely the double quotes surrounding reported speech and the parentheses, were concluded to lie outside the realm of the expressive power afforded by finite-state grammars. It was also concluded that a limited extent of satisfactory analysis would in theory be possible. This hypothesis was explored and concluded to result in very limited gain in exchange for large amounts of time-consuming, error-prone work whose ultimate impact on overall grammar performance was very hard to predict. It was hence

concluded that the implementation of a fix for these two linguistic features was unworthwhile.

The remaining constructs identified for potential fixing were addressed in turn. For each of them, we proposed and discussed a solution and described the details of its implementation. Some of these solutions were more involved than others, some requiring the addition of a single line to one file, some requiring changes in the workings of minor parts of the grammar and modifications to both the tagfixes and the grammar files.

A set of fifty sentences was picked from the corpus. Each sentence contained at least one word or symbol whose occurrence could inequivocally be assumed to indicate the presence of a problematic grammatical feature identified as explained above. Care was taken to provide a more or less equal number of sentences containing each of these words.

A golden standard was compiled by hand. This was done by taking a copy of the file containing the fifty selected sentences, and parsing them by hand to model our idea of a desirable, useful parse for each sentence.

The changes proposed for the grammar were implemented. The tagfixes and grammar files were compiled using the utilities specific to each purpose. A file containing unparsed versions of all the fifty selected sentences was created and the new file was parsed with the new grammar.

This set the stage for the beginning of the testing step *per se*. The Parseval metric was implemented and run on the three files: the golden standard, the fifty sentences as extracted from the original file (pre-edition corpus) and the same fifty sentences parsed by Cass with our new grammar (the post-edition corpus). The metric compared each of the pre- and post-edition corpora to the gold standard and thus came up with the numeric figures for precision, recall, mean and zero crossing brackets that we decided to use as an indicator of how satisfactory our modifications were.

Parseval was picked over a number of other known parser evaluation techniques by reviewing existing literature on parser evaluation and weighing the relative advantages and disadvantages of each scheme. After discussing each of the most popular parser evaluation metrics, Parseval clearly stood out because of its adequacy for the purposes of our project, and because of its simplicity of implementation and interpretation.

The numbers obtained in this step spoke quite clearly: mean precision went from approximately 0.71 to approximately 0.77. With mean recall, something similar happened: the pre-edition corpus displayed a precision measure of roughly 0.73, which saw itself boosted to almost 0.79 in the post-edition corpus. As for the crossing brackets, the mean number

of them per sentence came down to 2.54 in the post-edition parses from 3.46 in pre-edition ones. And the number of sentences with no crossing brackets went up by five: 8 before grammar modification, 13 afterwards.

The numbers are very encouraging, but there are still a number of problems. While the overall figures clearly spell an improvement, a significant number (eleven) of sentences saw their numbers go in the opposite direction of what we desired. Future work would focus on finding the causes of such unwanted behaviour. Also, the linguistic constructions spotted as problematic are but a few examples among many others which are not being treated as desired. For an improved fit of the grammar to the data, work should be done on finding more constructions not properly reflected in the grammar, assessing their prevalence in the data, and introducing changes to allow for them. A major redesign of the grammar might also be worth considering, rather than minor adjustments of the existing one, as was done for this project.

Appendix A

Listing of the Test Corpus

This is a list of the fifty sentences which were parsed by hand in the golden standard, with the original grammar in the pre-edition corpus and with the modified grammar in the post-edition corpus.

1. 4/88, I DEPARTED BOEING FIELD.
2. AFTER RECEIVING MY TRANSPONDER CODE, I LEVELED OFF AT 1500'.
3. I INFORMED ATC THAT I WOULD BE DSNDING TO AL ALT OF 500' WHEN I REACHED THE WATER.
4. I OBSERVERD IN MY ALTIMETER AN ALT OF APPROX 500' JUST N OF DAWAMISH HEAD.
5. AFTER THE PASS BY ALKIA, I DID NOT GO OVER LAND UNTIL I CLBED TO APPROX 1200-1300' PRIOR TO ENTERING THE TFC PATTERN AT BOEING FIELD.
6. WHEN I LANDED, TO MY SURPRISE, THE POLICE WERE WAITING FOR ME.
7. THE RPT MADE BY THE POLICE OFFICER SAID THAT I WAS AT AN ALT OF 35-50', WHICH IS FALSE.
8. WHEN THE ACFT WAS FUELED THE FUELER STATED THE FUELING PANEL GAUGES OPERATED NORMALLY AND ALL GAUGES READ ACCURATELY.

9. I THEN LOOKED UP AND ASCERTAINED THAT I HAD PARTIALLY ENCROACHED UPON THE ACTIVE RWY AND OBSERVEd ACR Y WITH LIGHTS ON APPROX 3000' DOWN RWY 1 ON TKOF ROLL.
10. WE WERE NEAR V 1 WHEN X ENTERED OUR RWY.
11. ROTATION WAS AT VR AND WE WERE AIRBORNE ABOUT 200' AS WE PASSED X, WHO WAS JUST CLRING OUR RWY.
12. THE WDB X WAS 300' HIGH AND CLBING AT ABOUT 1500 FPM.
13. WHEN THE THRUST LEVER WAS ADVANCED, THE QUANTITY INCREASED TO 1.7 GALS AND STAYED FOR QUITE A LONG TIME.
14. ATC GAVE US A BLOCK ALT OF FL 290/310.
15. ACFT AT THE GATE WAS FACING 070 DEGS WITH ATIS WINDS 220/17 G 30 (TAILWIND).
16. #2 ENG START WAS ABORTED WHEN IT APPEARED OF A POSSIBILITY OF A HOT START.
17. ALL PAX EVACUATED LEFT SIDE OF AIRPLANE (AWAY FROM THE FIRE).
18. WIND WAS 90/8 KTS.
19. I WOULD ESTIMATE THAT HE WAS 350-400' LEFT OF CENTERLINE WHEN I FIRST SAW HIM.
20. OR, WHEN CLRED TO BACKTAXI 17" WOULD SUFFICE.
21. THIS SHOULD NOT HAVE HAPPENED (THEY ENDED UP SIDE BY SIDE 1 1/2 MI FINAL).
22. WHEN SMT Y BROKE OUT HE PROBABLY SAW APPCH LIGHTS OFF HIS LEFT AND WENT FOR THE RWY.
23. W AND E LCL ALSO HAVE RADAR IN THE TWR, BUT TOOK NO ACTION TO PULL EITHER ACFT OFF THE APCH WHEN SEP DECREASED.

24. THE ACFT INVOLVED WERE A CONSIDERABLE DISTANCE AWAY FROM EACH OTHER WITH INCREASING SEPARATION.

25. THE SMA DEPARTED AND TURNED SLIGHLY AWAY FROM US.

26. HUMAN FACTORS: I WAS PICKING UP MY BEST MAN FOR MY WEDDING FOR 3/SAT/88.

27. NORMALLY DON'T FLY INTO TWR AIRPORTS (LAST TIME WAS 7/87), BUT HAD A LOT OF THINGS ON MY MIND AND THINGS TO DO.

28. HE WAS NOT AWARE THAT THE CTL FAC (MCC) FOR DEP CTLR WAS MANY MILES AWAY AND WAS CONFUSED WHEN HE COULD NOT CONTACT DEP CTL.

29. I DON'T RECALL EXACTLY THE HORIZ DISTANCE, BUT I BELIEVE IT WAS 2 1/2 - 3 MI.

30. I PREFER TO VIEW THIS INCIDENT AS AN ABORTED TKOF/CLBOUT AND A PRECAUTIONARY RETURN FOR LNDG WITH A STEADY GREEN LIGHT LNDG CLRNC.

31. HE ALSO ADDED THAT WE STILL HAD 20 MILE TO GO BECAUSE OF MANY ARRIVALS.

32. ALTHOUGH WE WERE DIVERGING COURSES, THE OTHER ACFT APPEARED TO BE 2-3 MI AWAY AND WE WERE TO BE CLBING THROUGH HIS ALT.

33. I KNEW OF TWO POWERLINES XING THE RIVER, BUT BOTH OF THEM WERE FAR AWAY FROM ME.

34. WE TALKED ABOUT THE HEARBACK PROB IN ATC, AND HE SAID THAT HE HEARS MANY PLTS AND CTLRS WHO DO NOT SEEM TO USE PROPER RADIO PROC AND PHRASEOLOGIES.

35. SUPPLEMENTAL INFORMATION IN ACN 84901: IN MY OPINION, THE CTLR WAS HANDLING TOO MANY ACFT AT ONCE.

36. IT IS USED ON MANY OLDER ACFT.

37. THE ACFT PASSED APPROX 2 1/2 MI APART WITH 600' ALT SEP.

38. DCA CTLR SAW ACR Y MODE C LEVEL AT 9000', SO GAVE TURN TO RPTR, ACR X, WHEN THE ACFT WERE STILL ABOUT 8 MI APART.

39. OFTEN TIME IN LAX, COMMUTER ACFT WERE EXPECTED TO TAKE TOO MANY CHANCES REGARDING WAKE TURB IN ORDER TO ALLOW APCH TO FIT MORE ACFT IN FOR ARR.

40. THE EXPERIENCE SHOULD HELP ME EXPLAIN TO MANY OTHER PLTS THE CORRECT PROC, BUT THESE CHARTS REALLY NEED A COLOR EMPHASIS OR CHANGE.

41. THERE WERE MANY OTHER ACFT ON FREQ, INCLUDING AN ACR FLT XBC.

42. WE DID NOT SEE ANY TFC TO OUR LEFT, BUT WE DID SEE AN ACFT AT 2 O'CLOCK, AT LEAST 2000' HIGH AND AT LEAST 5 MI AWAY, HDG SBND.

43. IT WAS TOO FAR AWAY TO IDENT THE TYPE.

44. DURING THIS WHOLE PERIOD OF TIME, THERE WERE MANY INSTANCES OF THE FREQ BEING BLOCKED BY SEVERAL ACFT TRANSMITTING AT THE SAME TIME.

45. WE WERE AT APPROX THE SAME ALT AND ABOUT 300' APART.

46. IFR FLT PLAN BETWEEN OPF AND TMB (18 NM APART, BOTH FIELDS WERE IFR) ON RADAR VECTORS (210 DEGS).

47. A VFR CLB WAS GIVEN TO THE ACR BECAUSE THE CTLR THOUGHT THE 2 ACFT WERE STILL VERY FAR APART AND WOULD NEVER BE A FACTOR FOR EACH OTHER.

48. AT THE CLOSEST POINT, THE ACFT WERE ONE MILE APART, SLOWLY DECREASING, AND SEPARATED BY 100' ALTITUDE.

49. SINCE THE 2 ARPTS ARE ONLY 10 MI APART, IT WOULD BE BARELY 15 FLYING MILES TOTAL, ALLOWING FOR A 4-5 MI FINAL.

50. AT INITIAL CALL UP, SMT X WAS TURNED 20 DEGS LEFT BY THE TRNEE (SMT X AND ACR Z WERE APPROX 45 MI APART).

Bibliography

[1] Steven P. Abney. Parsing by chunks. In Steven Abney Robert Berwick and Carol Tenny, editors, *Principle-based Parsing*, pages 257–278. Kluwer Academic Publishers, 1992.

[2] Steven P. Abney. Part-of-speech tagging and partial parsing. In Steve Young and Gerrit Bloothooft, editors, *Corpus-Based Methods in Language and Speech Processing*. Kluwer Academic Publishers, 1996.

[3] Steven P. Abney. The scol manual. `http://www.ims.uni-stuttgart.de/~light/introclss97/scol.ps`, 1997.

[4] Aviation safety reporting system. http://asrs.arc.nasa.gov/.

[5] Srinivas Bangalore, Anoop Sarkar, Christy Doran, and Beth-Ann Hockey. Grammar and parser evaluation in the xtag project. In *Workshop on Evaluation of Parsing Systems*, 1998.

[6] John Carroll, Ted Briscoe, Nicoletta Calzolari, Stefano Federici, Simonetta Montemagni, Vito Pirrelli, Greg Grefenstette, Antonio Sanfilippo, Glenn Carroll, and Mats Rooth. Shallow parsing and knowledge extraction for language engineering. http://www.ilc.cnr.it/sparkle/wp1-prefinal/wp1-prefinal.html, 1996.

[7] John Carroll, Ted Briscoe, and Antonio Sanfilippo. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*, pages 447–454, 1998.

[8] John Carroll, Annette Frank, Dekang Lin, Detlef Prescher, and Hans Uszkoreit. Beyond parseval: Towards improved evaluation measures for parsing systems. http://let.dfki.uni-sb.de/BeyondPARSEVAL/, 2002.

[9] Kenneth W. Church. *On Memory Limitations in Natural Language Processing*. Indiana University Linguistics Club, 1980.

[10] Kenneth W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, pages 136–143. Association for Computational Linguistics, 1988.

[11] Ronald A. Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, and Victor Zue. Survey of the state of the art in human language technology. http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html, 1996.

[12] Christine Doran. Punctuation in quoted speech. In *Proceedings of SIGPARSE 1996: Punctuation in Computational Linguistics*, pages 9–18, 1996.

[13] Eagles: Expert advisory group on language engineering standards. http://www.ilc.cnr.it/EAGLES96/home.html.

[14] Zellig Harris. *String Analysis of Sentence Structure.* Mouton, The Hague, 1962.

[15] Daniel Jurafsky and James H. Martin. *Speech and Language Processing.* Prentice Hall, 2000.

[16] Catherine Macleod Ralph Grishman and John Sterling. Evaluating parsing strategies using standardized parse files. In *Proceedings of the Third ACL Conference on Applied Natural Language Processing*, pages 156–161, 1992.

[17] Geoffrey Sampson and Anna Babarczy. Limits to annotation precision. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, pages 61–89, 2003.

[18] Beatrice Santorini. Part-of-speech tagging guidelines for the penn treebank project (3rd revision, 2nd printing). Technical Report, Department of Computer and Information Science, University of Pennsylvania, 1990.

[19] University centre for computer corpus research on language. http://www.comp.lancs.ac.uk/computing/research/ucrel/.