Visualising the Impact of Changes to Precision Grammars

A thesis presented
by
Ned Letcher
to
The Department of Computer Science and Software Engineering
in partial fulfillment of the requirements
for the degree of
Bachelor of Science (Honours)

University of Melbourne
Melbourne, Australia
November 2010
Visualising the Impact of Changes to Precision Grammars

Abstract
The development of precision grammars is an inherently resource intensive process. In this thesis we investigate approaches for providing grammar engineers with greater feedback on the impact of changes made to grammars. We describe two different visualisations which are created by comparing parser output from two different states of the grammar. The first involves the ranking of features found in parser output according to their magnitude of change so as to provide a low-level picture of the affected parts of the grammar. The second involves performing clustering over sentences whose parsability has changed in an attempt to find related groups of changes and accompanying sentences which exemplify each locus of change. These approaches provide complimentary avenues of feedback which can hopefully improve the efficiency of the grammar engineering development process.
# Contents

Title Page ................................. i  
Abstract ................................ iii  
Table of Contents ........................ iv  
Acknowledgments ......................... v  

1 Introduction ............................ 1  

2 Background ............................. 3  
  2.1 Precision Grammars .................... 3  
  2.2 Grammar Engineering .................. 4  
  2.3 A Gap in the Toolchain ................. 6  
  2.4 Previous Work ........................ 7  
  2.5 Filling the Gap ....................... 9  

3 Methodology ........................... 11  
  3.1 Resources ........................... 11  
  3.2 Preliminary Steps ..................... 11  
  3.3 Feature Ranking ..................... 14  
  3.4 Item Clustering ...................... 17  

4 Evaluation ............................ 22  
  4.1 Experimental Setup ................... 22  
  4.1.1 User Annotations .................... 22  
  4.1.2 Grammar Breakages ................. 24  
  4.2 Results and Discussion ............... 25  
  4.2.1 User Annotations .................... 25  
  4.2.2 Grammar Breakages ................. 26  

5 Conclusion ............................ 29  
  5.1 Concluding Remarks ................... 29  
  5.2 Future Work ........................ 30
Acknowledgments

Thanks to my supervisors, Tim Baldwin and Rebecca Dridan, for so generously giving much of their time towards this project. Thanks also to Emily Bender, Francis Bond, Dan Flickinger and Chikara Hashimoto for their assistance.
Chapter 1

Introduction

In this thesis we investigate techniques for generating visualisations of the impact of changes made to precision grammars. These visualisations are intended to combat one of the most significant drawbacks of linguistically accurate hand-crafted grammars — the considerable amount of time and resources required to create and improve their coverage. This oft-cited difficulty with deep parsing approaches (Abney (forthcoming), Baldwin et al. (2007), Zhang and Kordoni (2006)) has been one of the catalysts for the rise in popularity of statistical or “shallow” parsing approaches which can be bootstrapped from existing corpora and scale easily with the addition of further data. Precision grammars, however, offer distinct theoretical and practical advantages over shallow parsing approaches. They yield richer linguistic representations and are able to analyse the semantic structure of more complex constructions. As applications such as information retrieval and machine translation place increasingly greater demands on natural language processing technologies, it seems likely that deep parsing approaches will have an increasing role to play and there is thus a need for techniques which help reduce the amount of time required for developing precision grammars.

The visualisations produced using the techniques developed in this thesis are intended for use during the grammar development cycle, to be presented to grammar engineers as feedback on the impact of a change just made to a grammar. The hope is that this feedback will assist grammar engineers in more readily understanding how the grammar has been affected by the change, thus reducing the amount of time required to track down unanticipated problems that may be associated with the change.

Previous work that has involved providing different types of feedback for use in the grammar engineering process (Oepen and Carroll (2000), van Noord (2004), Waterman (2009)) has largely focused on the analysis of parser output from just one state of a grammar. Our approach is instead to look at two states of a grammar: one from before and one from after a change made to a grammar, focusing on ways of highlighting the differences between the two. One way of conceptualising this work
is as an investigation into techniques for the creation of a diff tool that takes as
input not source files, but parser output from running two different versions of a
grammar across the same corpus. Rather than trying to improve upon or replace
existing development tools used by grammar engineers, we see this tool as being
complementary, filling a currently existing gap in the grammar engineering toolchain.

We focus on two distinct types of visualisations, each corresponding to separate
use-cases of the tool in the grammar engineering process. The first involves a low-level
analysis which ranks features from the parser output in descending order of the degree
to which they are deemed to have been associated with the change. This is intended
to provide a broad picture of how the grammar has changed, allowing the grammar
engineer to see at a glance whether components of the grammar have actually been
modified as intended, and whether any components not intended to be modified
have been significantly affected. The second type of visualisation involves performing
clustering over the changed items from the corpus, with each cluster representing a
locus of change in the grammar. By presenting representative items closest to the
centres of each cluster this approach provides the grammar engineer with an idea of
different patterns of changes that may have occurred.

We focus on grammars developed within the DELPH-IN community, in particular
the English Resource Grammar (Flickinger 2009) and the Jacy Japanese Grammar
(Siegel 2000). While the techniques we developed were implemented for use with
the DELPH-IN grammar engineering environment, we believe that these techniques
could be applied to other precision grammar frameworks.

The remainder of this thesis is divided up as follows. Chapter 2 expands upon
the concept of a precision grammar as well as what is involved in the task of gram-
mar engineering so as to motivate the work in this thesis. Chapter 3 outlines our
methodology, including the resources we used and the steps taken to create the vi-
ualisations. Chapter 4 outlines the evaluation of the techniques introduced in this
thesis, including the data used and the experimental setup, and a discussion of the
results. Chapter 5 concludes the thesis as well as looking at further work that could
be done in this area.
Chapter 2

Background

2.1 Precision Grammars

Precision grammars are motivated by the desire for automated and accurate linguistic analysis of natural language. They are usually based upon or heavily influenced by formal theories of syntax developed within the field of linguistics. The syntactic theory that informs the DELPH-IN grammars used in this thesis is the Head-Driven Phrase Structure Grammar (HPSG: Pollard and Sag 1994) framework. Another notable syntactic framework that is used for the development of precision grammars is the Lexical Functional Grammar (Dalrymple 2001) framework, which is the basis of grammars developed in the ParGram Project.

These foundations in formal syntax mean that precision grammars all share the property that they draw a sharp distinction between sentences which are grammatical and those which are not. Compared with conventional grammar induction, precision grammars also generate much more detailed analyses and are able to capture much more complex linguistic constructions. In addition to producing syntactic analyses, precision grammars also generate detailed representations of the underlying semantic structure of language. It is for these reasons that the use of precision grammars to perform natural language parsing is often referred to as deep parsing.

The advantages of deep processing approaches are considerable. From a theoretical perspective, linguists have much to gain from the existence of high-quality precision grammars. Bender (2008) highlights two advantages of using broad-coverage precision grammars for the task of syntactic hypothesis testing. One is that the task of analysing a sentence in a given syntactic formalism is generally quite time consuming, making it a prime candidate for automation. Advances in parsing technologies now mean that complete analyses of sentences are possible in real time, and test suites of thousands of sentences can be processed within minutes. When this is coupled with the sizable corpora of natural language that are now available, it is clear that a syntactic hypothesis tested with the aid of precision grammars is going to have a stronger case for validity. The second advantage is that they allow for the holistic testing of
formal linguistic frameworks. These frameworks tend to be quite modular with, for example, the phonology of a language being handled by a separate component from that of the syntax. These components are often only tested in isolation when being developed. The construction of an end-to-end parsing system based upon a complete linguistic framework forces these components to be tested in unison, potentially revealing unexpected effects in the way different components of the framework behave together.

Another form of hypothesis testing in linguistics is cross-linguistic hypothesis testing. This type of investigation — generally referred to as linguistic typology — can be greatly aided by precision grammars, this time through the use of parallel grammars. These are sets of grammars of different languages that are not only built on the same syntactic framework, but also share a common strategy towards linguistic representation and analysis. There are two large-scale parallel grammar projects in existence, the Grammar Matrix project (Bender et al. 2002), based in the DELPH-IN community, and the ParGram project (Butt et al. 2002).

There are also applied uses for precision grammars in many technologies that leverage natural language processing. Some examples include information retrieval (Bobrow et al. 2007), machine translation (Oepen et al. 2004), and grammar checking (Crysmann et al. 2008). In general, any task which is aided by enhanced representations of the meaning of language can potentially benefit from deep parsing approaches. It is certainly true that statistical approaches to natural language parsing have proven successful to date, but as the demands for increased accuracy and better extraction of semantic information grow stronger, deep processing techniques are proving more and more useful.

2.2 Grammar Engineering

Grammar engineering is the process of developing and extending the coverage of precision grammars. It is a resource intensive process, involving numerous grammar engineers who require detailed linguistic training as well as an understanding of the framework in which the grammar has been implemented. A number of factors combine to make grammar engineering a slow and consuming process. One is the considerable size and complexity of these grammars. The most recent incarnation of the English Resource Grammar,\(^1\) for example, contains around 37,000 lexical items, 4200 types, and 275 rules (Flickinger 2009). These components involve a high degree of interaction between each other, meaning that modifications made to one part of the grammar can often produce flow-on effects entirely unexpected by the grammar engineer. The other factor is the high degree of accuracy that is demanded of these grammars. Over time, as improvements are made and the grammar is expanded to handle more linguistic constructions, it is vital that the linguistic validity of analyses produced by

---

\(^1\)The Barcelona release, July 2009
the grammar does not degrade, and that the grammar does not stop being able to parse linguistic constructions that it once could.

Just as in software engineering, the grammar engineer has a number of tools available to provide quality assurance. In grammar engineering the practice of monitoring the quality of the analyses produced by the grammar is referred to as grammar profiling. This involves the maintenance of test suites which facilitate the monitoring of the accuracy and coverage of the grammar, providing a means of ensuring that components of the grammar do not regress. In the DELPH-IN community these test suites are referred to as profiles and the individual tests in a profile are referred to as items. This terminology will be used throughout this thesis.

Perhaps the most prominent aspect of the grammar that must be monitored is the range of a language that it is able to handle — the coverage of the grammar. This is monitored by profiles constructed to track the extent to which a grammar undergenerates, which occurs when it does not provide an analysis for an item that is actually grammatical. Undergeneration usually occurs due to missing entries in the lexicon or gaps in the coverage of the linguistic constructions. It is also possible for grammars to be too permissive and provide analyses for items which are actually ungrammatical. When this occurs the grammar is said to overgenerate. This can be tested via the use of profiles containing negative items of ungrammatical text.

Another aspect of the grammar that must be monitored is the amount of ambiguity found in the grammar. It is normal for precision grammars to produce more than one analysis for an item, and quite often the number of analyses produced explodes to above 500 or even 1000. While it is true that ambiguity is simply a fact about language and can never be eliminated, it is desirable to remove excessive ambiguity where possible, provided that the modifications remain faithful to the underlying linguistic structure. Thus grammar profiling tools also report the number of analyses generated for each item. They also provide information regarding any changes in the operational performance of the grammar, such as the time taken to process profiles and the number of operations required for parsing. Just as with testing in traditional software development, any unexpected changes in any of these aforementioned statistics are potentially symptoms of introduced errors.

A limitation of grammar profiling tools is that on their own they are not able to detect the misanalysis of items. This occurs when an item has been successfully parsed but for the wrong reasons; its analysis does not match its actual syntactic structure. The devices used in conjunction with grammar profiling tools to monitor the accuracy of analyses are treebanks. These are corpora of items accompanied by a set of analyses elicited from the grammar which are deemed to be accurate by an expert annotator. Whenever a change is made to the grammar that alters the analysis of any of the items which are being treebanked, the grammar engineer can then check to see that the change has improved the analysis rather than making it worse. This approach also allows for the construction of regression tests, small profiles of treebanked items containing constructions whose analysis is considered final and
should never change.

A significant problem with the use of treebanks, is that they are time-consuming to annotate and usually need to be created from scratch for a new domain. Furthermore, the analyses must be kept up to date with any changes made to the grammar so that they continue to reflect the best analysis that the current state of the grammar can provide. This means treebanks must always be manually updated after any change is made to a grammar that affected the analysis of any items. Given that there can often be hundreds of changed items, this is a slow and labour-intensive process, adding considerably to the amount of time required for a change to be added to a grammar. It is however a necessary part of the grammar engineering process if the accuracy of the analyses produced by the grammar is to continually improve.

2.3 A Gap in the Toolchain

Grammar profiling and treebanking are both vital to the grammar engineering process, and represent opposite extremes in the level of feedback they provide. On the one hand, grammar profiling gives immediate but very broad feedback regarding the coverage and performance of the grammar. On the other hand, treebanking provides fine-grained feedback on an item-by-item basis of how the grammar was affected by a change. This detailed level of analysis is not only crucial to catching problems not visible in the grammar profiling stage but also for simply building up a picture of how the analyses produced by the grammar have changed with the modifications to the grammar. The problem is that the manual inspection and updating of treebanks is a slow process. The complexity of these grammars means that modifications in one area of the grammar can often have unexpected flow-on effects in other areas, and an all too real possibility for the grammar engineer is locating a serious problem in a treebanked item after considerable time has already been invested in manually analysing items. At this point the grammar engineer must correct the problem and start the process all over again.

The problem is that there is no middle ground between these two levels of feedback. The grammar profiling stage can often locate glaring problems such as coverage levels suddenly dropping or not changing at all when expected to, but often the real nature of the impact of the change is hidden to the grammar engineer; only via the slow inspection of treebanks is it slowly illuminated. It would be highly desirable for there to be tools which assisted the grammar engineer in more quickly acquiring an understanding of how the grammar has been affected. Such a tool would not aim to help the grammar engineer understand the complete impact of a change – this would be an unfeasible goal – instead it would aim to simply make the impact of the change more transparent. This is the aim of the visualisations presented in this thesis.
2.4 Previous Work

This section outlines previous works in the literature which provide techniques for profiling the current state of a grammar. These represent the range of different tools or approaches that the grammar engineer has available in order to gain a better understanding about the current state of the grammar. Each approach is assessed on the basis of how well it could be used in the context of providing feedback on the impact of a change just made to a grammar. In particular, the kind of feedback that is desired can be broadly encapsulated by these two questions:

1. Were any errors inadvertently introduced by the change?
2. Did the change actually have the intended effect?

One approach towards gaining feedback on the current state of the grammar is that of Baldwin et al. (2005). In order to build up a picture of the coverage of the English Resource Grammar over the British National Corpus, they manually profile the parse results, identifying all the reasons for parse failure across the whole corpus. This resulted in a taxonomy of different types of parse failure, allowing them to gauge the different weaknesses of the grammar. While this technique is excellent at uncovering the specific weaknesses a grammar, the resources required for such an exercise mean that it cannot be performed on demand, and is thus unsuitable for use in gaining feedback on a change to the grammar.

The tool that performs the task of grammar profiling for grammars produced in the DELPH-IN community is a software package called [incr tsdb()] (Oepen and Carroll 2000). It facilitates the construction of profiles designed to test for overgeneration and undergeneration in the grammar as well as changes in the number analyses produced per item. It also provides an environment for carrying out the treebanking process. The grammar engineer uses this tool by first inspecting the high-level results, looking to see if there are any obvious problems with the changes made, before beginning the process of analysing the changed items in the treebanks to ensure that that changes to the analyses were beneficial. [incr tsdb()] is able to be used in a comparative mode, directly comparing the coverage and performance of two different versions of a grammar. This makes it a valuable tool for gaining feedback on the nature of a change just made to a grammar, however it suffers from the previously mentioned limitations of grammar profiling tools — that this information only offers a limited degree of feedback on the nature of the change made; any further information requires the laborious inspection of changed items within the profiles.

One approach that could be used specifically for the detection of introduced problems into the grammar is the error mining technique of van Noord (2004). This approach involves dividing the corpus up into items which yield successful parses, and those which do not. A metric for measuring the parsability of $n$-grams (sequences of tokens of length $n$) is devised, which is taken to be the ratio of the number of times
the \( n \)-gram occurs in successfully parsed sentences over the number of times it occurs in the whole corpus. The parsability of variable length \( n \)-grams is calculated for all sequences that occur above a fixed frequency cut-off in the set of failed items. This list is then ordered, with the entries with the lowest parsability values representing sequences most indicative of problems in the grammar. This technique has become known as grammar error mining. A limitation of this approach is that the absolute frequency of word sequences in the corpus is not taken into account in the parsability metric, which has the effect of giving sequences with lower frequencies a less reliable parsability score.

A further problem with the technique of van Noord (2004) is that the parsability metric does not take into account the “suspiciousness” of other sequences in the item, leading to sequences receiving disproportionately low parsability scores simply for occurring alongside a strongly suspicious sequence. This motivates the work of Sagot and de La Clergerie (2006), which replaces the ratio metric of van Noord, with one defined by the mean probability of a sequence being the source of a problem across all occurrences of that sequence in the corpus. The calculation of this relies on the assumption that there is only one source of error for an unparsable item. Since the probability of a sequence being the cause of an error in a item depends on the probabilities of other sequences in the item, this function is recursively defined. In their implementation, the algorithm used is applied iteratively until a point of convergence is arrived at, which has led to it being referred to as iterative mining. A limitation of this iterative error mining approach was that \( n \)-grams with \( n > 2 \) lead to data sparseness problems. A further problem with including longer \( n \)-grams identified in the work of de Kok et al. (2009), is that they can mask the suspicious nature of smaller \( n \)-grams occurring within them. They introduce a technique for including longer \( n \)-grams by adding a preprocessing module which expands each \( n \)-gram on the condition that it is more suspicious than each \( n - 1 \)-gram within it.

One of the advantages of grammar error mining is that the only input required is a binary success or failure value for each item, meaning that they are agnostic towards the implementation of the parser or the grammar. This simplicity arises from the fact that the only assumption made is that a failed parse is an indicator of suspicion for the sequences of words within that item. This is not always true, however, as in the case of ungrammatical items, whose unparsability is entirely suspicion free. A further consequence of the aforementioned assumption is that successful parses are not considered indicators of suspicion. This is not a always a valid assumption, as in the case of overgeneration. Another common problem with precision grammars is that items can be misanalysed — they are given a successful parse but with an incorrect analysis. Once again, error mining cannot locate these problems. Consequently, error mining techniques are only able to locate errors of undergeneration.

A different approach to error detection in precision grammars is that of Goodman and Bond (2009). They leverage the ability of precision grammars to both generate as well as parse text. The general approach is to take the underlying semantic form of
a item that has been parsed and use it to try to generate the original surface form of the item. If this process fails then there is a likely problem with the grammar, since precision grammars are in principle symmetric with respect to parsing and generation. Instead of $n$-grams from the surface strings of these items, $n$-grams of paths in the parse tree are used, which results in a ranking of rule paths in the grammar — rather than word sequences — according to their likelihood of being problematic. This is advantageous as it gives the grammar engineer a more detailed window into the context of the problem as it appears in the grammar. As with error-mining, this also has the limitation that it can only locate problems of undergeneration in the grammar.

These different techniques for automated error detection are useful for directing the development of a grammar, as they are good at locating gaps in coverage. They are ill-suited, however for providing feedback on immediate changes made to a grammar, as there is no guarantee that the problems they report will have anything to do with changes just made to the grammar. A further issue with the error mining technique approach (although not with the approach of Goodman and Bond (2009)) is that the results are presented in terms of word sequences. When seeking feedback on a change made to the grammar, the grammar engineer seeks to know how their change has affected the components of the grammar that were explicitly modified as well as any other components they may have interacted with. This is difficult to ascertain from the output of error mining tools since the output is in terms of word sequences whose underlying form in the grammar are not known concretely, since they originate from items which did not receive an analysis.

A tool which can shed light on the impact of changes made to a grammar, is the Oceanography tool (Waterman 2009) from the XLE grammar development environment. Oceanography works by processing parser output and collating information such as counts of constructions as well as other statistics along with the context in which they occurred. Dost and King (2009) describe making use of Oceanography to gauge the impact of modifications made to the grammar. To confirm that a change has been applied successfully they suggest comparing the relative frequencies of the modified constructions between the initial version of the grammar and the modified version. Sudden changes in frequencies of constructions serve as a symptom of introduced problems. Using Oceanography in this comparative manner is similar to our proposed approach, however we investigate the potential for a single unified output that focuses on capturing the nature of only the modifications made to the grammar.

### 2.5 Filling the Gap

Grammar error-mining and the detailed parse result profiling of Oceanography come the closest, however both fall short of meeting the objectives of providing a visualisation specifically geared towards providing feedback on the nature of a change
made to a grammar. In the case of error-mining it is unsuitable for providing feedback on a specific change, and in the case of Oceanography it does not facilitate the direct comparison of two different grammars, instead profiling a single state of a grammar.

In order to try to fill this gap in the grammar engineering process, we came up with two different techniques for generating visualisations of changes made to precision grammars. Like the parse-result mining of Oceanography, these approaches involve processing the output produced during the parsing process. Our approaches diverge from Oceanography in that visualisations are generated by comparing the output of two different versions of a grammar rather than just one version. Unlike grammar error-mining techniques these visualisations do not assign judgements such as “suspiciousness” to results, but instead simply provide a sufficient amount of information from which the grammar engineer can draw their own conclusions.

The two different types of visualisations correspond to two different use-cases in the grammar engineering process. The first type of visualisation is a simple ranking of features found in the parser output, sorted by the degree to which they were affected by the change made to the grammar. This visualisation is intended to provide the grammar engineer with a broad sense of how the components of the grammar have been changed. It is hoped that by looking at the top ranking features, the grammar engineer would be quickly able to confirm that components of the grammar intended to be changed have actually been affected, as well as identifying unexpected features which may be symptomatic of an introduced error.

The second type of visualisation involves the presentation of a number of items automatically selected from the corpus via clustering techniques. These different items are intended to be indicative of different types of changes identified in the parser output. This type of visualisation is intended to provide a more nuanced picture of the impact of a change to the grammar, which would more likely be used after the grammar engineer has inspected the feature ranking visualisation.

Rather than trying to improve or replace any of the existing tools and approaches mentioned above, we see these approaches as being complementary, representing new additions to the grammar engineer’s toolchain with distinct use-cases from the existing tools.
Chapter 3

Methodology

3.1 Resources

For this investigation we used Jacy, a Japanese grammar (Siegel 2000), and the English Resource Grammar (ERG: Flickinger 2009), both HPSG based precision grammars developed within the DELPH-IN community. For our test profiles, we selected corpora used in the development of the grammars. In the case of Jacy this is the Tanaka Corpus (Tanaka 2001) and for the ERG this was a selection of different corpora found in the Redwoods Treebank (Oepen et al. 2002). Since this investigation revolves around changes made to these grammars, we also needed a source of modifications made to these grammars. Instead of creating our own changes from scratch, we used real changes made in the course of the development of these grammars. This ensured that we would be working with sample changes of the sort that would actually occur during the course of the grammar engineering process. The changes we used are the same that were used for the evaluation and are outlined in Chapter 4.

3.2 Preliminary Steps

For each change to a grammar being investigated, we began by taking two versions of the grammar: the initial state of the grammar and the modified state of the grammar. These were then both used to parse the same corpus. As numerous changes were investigated over a number of different corpora, the running time of the parser quickly became an issue. To address this we restricted the maximum number of analyses that the parser returned for each item. This is possible because PET, the DELPH-IN parser, employs a parse selection model (Toutanova et al. 2002), which assigns a probability distribution to the different analyses produced for a given input in accordance with the likelihood of analyses being preferred. Limiting the number of analyses to be found improves the parsing speed due to the selective unpacking (Zhang et al. 2007) that is used by PET. This sees the full set of analyses yielded by
Figure 3.1: Derivation tree produced by the ERG for the sentence *The dog sat*.

dog_n1 := n._c.le &
[ ORTH < "dog" >,
SYNSEM [ LKEYS.KEYREL.PRED ".dog_n.1.rel",
PHON.ONSET con ] ].

Figure 3.2: Lexical entry for the word *dog* in the ERG.

a grammar for an item being stored in the one graph (referred to as a *parse-forest*) with alternative subnodes being packed together. By unpacking these alternatives from the parse-forest on an as-needed basis, a significant reduction in parsing time can be had if a limit is placed on the number of analyses to be found. For Jacy and the ERG it is common for PET to locate over 500 analyses for a sentence. We restricted the parser to returning only the top ten analyses for an item, which yielded a considerable reduction in parsing time.

We then created two sets of items for each grammar pair: one containing the items which previously parsed but after the modifications did not, and the other containing items which previously did not parse but did after the change. For simplicity, we excluded those items which previously parsed and continued to do so, as these items yield parser output corresponding to both versions of the grammar which serves to complicate comparison with items receiving an analysis from only one version of the grammar. These two categories, *parse* → *no parse* and *no parse* → *parse*, form the basis of both types of visualisations that we generate, with separate visualisations being produced for each category, as they reveal different kinds of effects resulting from the change to the grammar.

As was shown in Chapter 2, in order to gain the kind of feedback we are after it is necessary to leverage more information than just the parsability of an item. Just as Oceanography does, we chose to look for further information in the output
Figure 3.3: The derivation tree from Figure 3.1 with lexical entries replaced by their lexical types.

that is produced by the parser for successful parses. PET produces a number of different outputs for successful parses, each containing different kinds of structures built up during the parsing process. For our investigation we used the full derivation trees generated by the parser. Each of these trees represents the complete syntactic analysis of the item that was parsed. Being based on the HPSG paradigm, which is a type based formalism, DELPH-IN grammars are implemented using typed feature structures. The nodes of the derivation trees derivation tree are these typed features structures (or just types). An example derivation tree for an English phrase parsed with the ERG can be seen in Figure 3.1. In this example, just the labels of the nodes are shown, which correspond to the names of the types.

Since it is the types that the grammar engineer is usually working directly with when making changes to the grammar, we extracted the names of the types from the derivation trees to be used as the features for the creation of the visualisations. In the case of the leaves of the tree, extracting the surface form of the word would have lead to data sparsity problems due to the infrequent occurrence of most words. This would have also likely occurred had we used the penultimate nodes of the trees, which correspond to the lexical entry for that word. Lexical entries are the objects that populate the lexicon, with each entry containing information about a word. Figure 3.2 shows the lexical entry for the word dog in the ERG. It is composed of a lexical type n\_c\_le, which contains the properties that are common to all countable nouns, and then other information particular to the word dog. We extracted the lexical type of each lexical entry in order to avoid problems of data sparsity, but also because it is the lexical type rather than the lexical entry that the grammar engineer would be more likely to be concerned with when making a change to the grammar. Figure 3.3 shows the derivation tree from Figure 3.1 with the lexical entries replaced by their lexical types.

Another issue that arose was how features should be extracted from items that

---

1For an introduction to typed feature structures see Copestake (2002)
yielded more than one analysis. For these items, which occur quite frequently, it was fairly straightforward to extract the union of the features across all of the analyses. Doing this for all analyses may not be desirable, however, since the probability assigned to some of the analyses by the parse selection model could be quite low, indicating that these are less likely to be correct analyses of the item. Given this, it is not always true that the more analyses included the better, since features found in unlikely analyses are not contributing towards a better understanding of how the grammar has changed. It is also not clear at which point analyses should be excluded from contributing their features. This issue is investigated further in Section 4.

3.3 Feature Ranking

Having extracted features from the derivation trees of parsed items from the two categories of parse $\rightarrow$ no parse and no parse $\rightarrow$ parse, we then needed a means to rank features by the degree to which they were affected by the change. In order to achieve this we applied a weighting function to each feature. One option would have been to simply weight features by the change in the number of items that they occur in between versions of a grammar. This however has the problem that it biases features which occur frequently. In the area of information retrieval, this problem is often overcome by using the inverse document frequency of features to offset the frequency of features across the document collection (Manning and Schuetze 1999). The inverse document frequency (IDF) of a feature is calculated as the number of documents in a collection divided by the number of documents that contain the feature. This has the effect of decreasing the score of features that occur frequently in a collection, and increasing those that don’t occur frequently.

The IDF score of feature $i$ in grammar $j$ is given by Equation 3.1. $P_G$ is the number of items that parsed in grammar $G$ and $p_{i,G}$ is the number of items parsed with grammar $G$ that feature $i$ occurred in; the denominator is offset by one to avoid zero-division errors when the feature does not occur any of the parsed items.

$$IDF_{i,G} = \log \frac{|P_G|}{1 + |\{p_{i,G} : f_i \in p_{i,G}\}|} \quad (3.1)$$

Weighting the features according to the change in IDF ensures that features are weighted by the extent to which they are affected by the change in the grammar and are not biased by the frequency of features in the results, as more frequently occurring features then require even greater rates of change in order to gain a significant score. This weighting function for a feature $i$ is given by Equation 3.2, where $G$ is the initial version of the grammar and $G'$ is the new version of the grammar. Later on when performing the clustering, we also found that it was necessary to further emphasise strongly changed features relative to those less affected, by squaring the difference in IDF values. This is discussed further in Section 3.4.
\[ W_i = |IDF_{i,G} - IDF_{i,G'}|^2 \]  \hspace{1cm} (3.2)

A further issue we discovered with this weighting function is that when the number of parsable items containing a given feature does not change between grammars, this feature still receives a non-zero weighting. This is due to the fact that even if the number of items that parse with a certain feature does not change, the total number of items parsing most likely will change between grammars, yielding a different IDF score for the altered version of the grammar and thus an overall weighting of greater than zero. This is undesirable since the goal of the weighting function is to represent the degree of change found in that feature and to assign values greater than zero to items that did not change would be contrary to the point of the weighting function. In particular, it would be quite confusing for the grammar engineer when inspecting such a feature in one of the visualisations. The solution we came up for this was to modify the way IDF was calculated to use the total number of items parsing across both versions of the grammar rather than just one. The modified IDF formula is given in Equation 3.3. Replacing the standard IDF calculation in Equation 3.2 with this modified version had the effect of giving features that occurred in the same number of items that parsed in both grammars a weight of zero, while leaving the relative weighting of all other features the same. The final weighting function is given in Equation 3.4.

\[ \text{modIDF}_{i,G} = \log \frac{|P_G| + |P_{G'}|}{1 + |\{p_{i,G} : f_i \in p_{i,G}\}|} \]  \hspace{1cm} (3.3)

\[ W_i = |\text{modIDF}_{i,G} - \text{modIDF}_{i,G'}|^2 \]  \hspace{1cm} (3.4)

Armed with a weighting function, we were then able to create the feature ranking visualisations. These were simply the top ten ranked features, listed in order along with their weights. Separate visualisations were created for the categories parse → no parse and no parse → parse. Tables 3.1 and 3.2 show feature rankings for these categories, respectively, generated from a change made to Jacy during the course of its development. This change involved extensive changes to the handling of verb phrases. Features that were explicitly modified in the changes made to the grammar are presented here in boldface, but were not distinguished in the actual visualisations.

Both tables show visualisations generated using the modified IDF weighting function as well as the naïve weighting function, which uses the change in number of items containing each feature. This illustrates how the IDF-based weighting function alters the ranking of features by giving less weight to features whose changes have more to do with the fact that they occur more frequently. At first glance, the frequency based weighting function appears to produce superior rankings, since in both cases more of the features actually modified by the grammar engineer have risen to the top of the ranking. This, however, does not follow from the intention of the visualisations.
Table 3.1: Feature ranking generated from items in the parse → no parse category of a change made to the Jacy grammar during its development. This change involved extensive changes to the handling of verb phrases. Features in boldface indicate that they were explicitly modified by the grammar engineer.

The visualisations are not intended to represent the actual changes made to the grammar — this can be achieved simply by performing a diff over the source files of the grammar. Instead, the visualisations are intended to capture the effects that the changes had in the parse results. Due to the complex interactions between the components of precision grammars, many of these effects will manifest themselves as flow-on effects in components not directly modified. This means that we should hope to see features in the rankings that were not explicitly modified by the grammar engineer. Any such feature that is also highly ranked is then of interest to the grammar engineer, as this is previously unknown information about the impact that the change had upon the grammar.

Evidence for the inferior quality of the frequency based weighting function can be seen in Table 3.1, with the utterance-root feature occurring in the tenth position.
### Table 3.2: Feature ranking generated from items in the \textit{no parse} \rightarrow \textit{parse} category of a change made to the Jacy grammar during its development. This change involved extensive changes to the handling of verb phrases. Features in boldface indicate that they were explicitly modified by the grammar engineer.

<table>
<thead>
<tr>
<th>Weight</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.257 \textit{caus-intrans-scope-passcmorph-end-lex}</td>
</tr>
<tr>
<td>2</td>
<td>3.787 \textit{garu-sbj-change-rule}</td>
</tr>
<tr>
<td>3</td>
<td>2.134 \textit{vn-light-rule}</td>
</tr>
<tr>
<td>4</td>
<td>1.995 \textit{caus-trans-obj-passcmorph-end-lex}</td>
</tr>
<tr>
<td>5</td>
<td>1.922 \textit{v8-c-minusshon-stem-lex}</td>
</tr>
<tr>
<td>6</td>
<td>1.396 \textit{adversative-trans-pass-passcmorph-end-lex}</td>
</tr>
<tr>
<td>7</td>
<td>0.463 \textit{ke-lexeme-infl-rule}</td>
</tr>
<tr>
<td>8</td>
<td>0.164 \textit{comp-prpstn-lex-questarg}</td>
</tr>
<tr>
<td>9</td>
<td>0.164 \textit{hes-lex}</td>
</tr>
<tr>
<td>10</td>
<td>0.164 \textit{pure-aspect-progressive-shon-c2-stem-lex}</td>
</tr>
</tbody>
</table>

This feature should not appear at all in the ranking, as it is the root node of a large class of analyses. This means that any change in this feature is indicative only of general increase or decrease in the overall number of sentences parsing. Comparing this with the IDF-based weighting for this category, it does not occur at all in the top ten features.

#### 3.4 Item Clustering

The aim of the second form of visualisation is to locate different patterns of change in the parse results via the presentation of individual items, each representing a different pattern of change. We treated this as a clustering problem, with each cluster
representing a different type of change. The clustering technique we used was \( k \)-means, a simple clustering algorithm which attempts to partition a set of observations into \( k \) clusters by placing each observation in the cluster with the nearest mean. As with the feature rankings, separate visualisations were created for the categories of \( \text{parse} \rightarrow \text{no parse} \) and \( \text{no parse} \rightarrow \text{parse} \).

\( k \)-means is an iterative algorithm, which works by initially populating each cluster with a single observation which becomes the centroid of that cluster. The remaining observations are then allocated to the cluster with the nearest centroid. The centroids are then updated to be the mean of all the observations in their respective clusters. This process is performed again, with all observations being allocated to the clusters using their updated centroids, before updating the centroids once again. This process of assigning observations to clusters and updating centroids is repeated until the clusters converge, when the assignment of observations no longer changes.

In order to perform clustering over the items in these categories, each item was converted into a feature vector in \( n \)-dimensional space, where \( n \) was the total number of features located in the parse results from both versions of the grammar. For each item its feature vector was populated by setting the position of a feature to zero if it did not occur in the derivation tree for that item, or by setting it to the value returned by the feature weighting function given in Equation 3.4 if it did occur.

We used Euclidean distance to calculate the distance from each point to its centroid. Another decision needed to be made when implementing \( k \)-means is how to initialise the centroids of the clusters. The simplest approach is just pseudo-randomly selecting \( k \) observations, which has the effect that execution of the algorithm over the same dataset is not guaranteed to converge on the same solution. This is because \( k \)-means is a form of expectation-maximisation algorithm, which means there is no guarantee that the final solution will be the global optimal solution. If a poor selection of initial seeds is made then the solution will be a local optimum, and may be a long way away from the global optimum. We tried to avoid the selection of a poor set of seeds by pseudo-randomly selecting the first point, and then selecting each other point to be the furthest point from the mean of the ones already selected, until \( k \) points are selected.

One of the drawbacks of \( k \)-means is that the value of \( k \) must be specified in advance. This posed something of a problem since each cluster is intended to represent a particular pattern of change and the grammar engineer has no way of predicting this number in advance. This problem with the algorithm is often solved by repeatedly performing the clustering across a range of values of \( k \) and then selecting the value which results in the best clustering. One way of doing this is by selecting \( k \) such that it minimises the combined sum of the squared errors across each cluster, since the goal of the clustering is to minimise the distance from each observation to its centroid. The problem is that this favours higher numbers of clusters and thus requires the introduction of a penalty for each increase in \( k \) to offset this bias.

We chose to evaluate the quality of the clustering using the Silhouette algorithm
(Rousseeuw 1987), which is used as a means of evaluating the fit of an observation within a clustering of a dataset. This technique involves the calculation of a single score for each observation which is a combined representation of the internal similarity with other observations in the same cluster and of the dissimilarity with all other observations outside the cluster. The formula for calculating the Silhouette score of an observation $i$ is given in Equation 3.5. $a(i)$ is defined as the mean distance from $i$ to all other observations in the same cluster and $b(i)$ is defined as the mean distance from $i$ to all observations in the cluster closest to the cluster $i$ occurs in. This produces Silhouette scores ranging from negative one to one, with negative one indicating poor clustering and one indicating good clustering. The average Silhouette score for a whole cluster can then be computed and in turn, the average across all clusters, resulting in a final value representing the quality of the overall clustering. An advantage of using this aggregate Silhouette value to select the best value of $k$ is that it doesn’t involve any tunable parameters that need to be refined for use in this domain. An artifact of the Silhouette algorithm is that it is undefined when $k$ is equal to one. We did not see this as a problem however, since the purpose in this type of visualisation is tracking down multiple changes in the output.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$  \hspace{1cm} (3.5)

The final clustering was then done by performing the clustering for all values of $k$ between 2 and 20 and selecting the clustering which resulted in the highest Silhouette score. As previously mentioned, we found that we had to fine-tune the feature weighting function developed in Section 3.3. Prior to introducing the exponent in the weighting function found in Equation 3.4, the values of $k$ selected using the Silhouette score were consistently much higher than expected given the changes that had been made to the test grammars. Analysis of the clustering output indicated that many of the clusters in these results were centring around relatively unimportant features given the nature of the change to the grammar. The clustering was correctly picking up on changes in features, however many of these changes were so small as to be irrelevant to the understanding of how the grammar had been changed. This seemed to indicate that there wasn’t enough separation between the weights being assigned to significantly changed features and features less impacted by the change. Squaring the weightings had the desired effect of separating out significantly changed features from less affected features and resulted in a more reasonable number of clusters being generated for each change. We kept this modification to the feature weighting for use in the feature ranking visualisations as well, partly to ensure that features were comparable across both visualisation, but also because we felt accentuating the weighting of more heavily affected features would assist the grammar engineer in focusing on the more important features in the ranking.

After the clustering was performed, the result was a set of clusters for each of the categories parse $\rightarrow$ no parse and no parse $\rightarrow$ parse. The closest item to the
centroid of each cluster was then taken as the item exemplifying the type of change captured in each cluster. On their own, these exemplar items would not necessarily be that informative to the grammar engineer. We identified three pieces of information that a grammar engineer would require to interpret these the results. The first is the properties of each item that contributed towards making it the exemplar of its cluster. We added this information to the visualisation by presenting alongside each exemplar, a ranking of features similar to the ranking described in Section 3.3. This provided the grammar engineer with a simple means to establish the nature of the change hypothesised by each cluster.

The second piece of information we identified was the quality of each cluster used to generate an exemplar. This is important as some clusters will be better centred around their centroids than others. In clusters where items fit poorly, the top ranked features found within the exemplar item may not actually occur that frequently within other items in the same cluster, meaning that these features are not actually indicative of the items in that cluster. In this case these features have low cohesion. Another symptom of poor clustering is when the top features of an exemplar have high overlap, when they are common to many items outside of that cluster, meaning that these features are not particularly identifiable as belonging to this cluster as opposed to others. In both these cases of features with low cohesion or high overlap, it is important that the grammar engineer knows that they should have a lower degree of confidence in the evidence that they provide. We addressed this problem by adding cohesion and overlap scores to each feature in the ranking. The cohesion of a feature in a given item was calculated as the percentage of items in the same cluster where that feature occurred in the feature ranking. The overlap of a feature in a given item was calculated as the percentage of items outside the cluster where that feature occurred in the feature ranking.

Finally, we also presented the number of items contained within each cluster so the grammar engineer could gauge the size of the purported change. An example clustering visualisation is presented in Table 3.3.
### clusters Cohesion (%) Overlap (%)

#### Cluster 1
- **彼はジョンと命名された**
- **Size:** 3
- 64.27 **aux-vend-rule**
- 22.60 **lightverb-pass-end-lex**
- 0.02 **sa-lexeme-infl-rule**
- 0.01 **head-specifier-rule**
- 0.00 **adv-p-lex-np-nonexh**

#### Cluster 2
- **必要なら、政府は、不動産業者に土地の価格を落とすよう要請するだろう。**
- **Size:** 6
- 4.32 **vn-light-obj-change-rule-noun**
- 2.13 **vn-light-rule**
- 0.01 **head-specifier-rule**
- 0.00 **hi-adj-s-rule**
- 0.00 **comma-vmod2-lex**

### (a) Clustering of no parse → parse

### Clusters Cohesion (%) Overlap (%)

#### Cluster 1
- **この件についてのお求めを了承します。**
- **Size:** 53
- 2.13 **vn-light-rule**
- 0.02 **vstem-vend-rule**
- 0.01 **head-specifier-rule**
- 0.00 **ki-lexeme-infl-rule**
- 0.00 **hon-prefix2n-lex**

#### Cluster 2
- **皆さんは、皮膚がんになる危険性が大いにあり、体を弱りきらせ、食料の乏しい。**
- **Size:** 1
- 6.26 **caus-intrans-scope-passmorph-end-lex**
- 2.13 **vn-light-rule**
- 0.02 **vstem-vend-rule**
- 0.01 **head-specifier-rule**
- 0.01 **comma-vmod2-lex**

### (b) Clustering of parse → no parse

Table 3.3: Clustering visualisations generated from the same change to the Jacy grammar used to generate the feature rankings in Tables 3.1 and 3.2.
Chapter 4

Evaluation

Evaluation of the techniques investigated in this thesis is not straightforward. This is largely due to the fact that these techniques do not attempt to produce any kind of measurable judgement of correctness, but rather produce an array of data intended to be interpreted by the grammar engineer. This is an inherently subjective quality, making it difficult to evaluate. Ultimately the best indicator of the success of these techniques is that grammar engineers are able to use them effectively for more rapidly gaining feedback during the grammar engineering process. This form of hands-on evaluation in the context of a real precision grammar project, was unfortunately not available to us. This chapter outlines the methods of evaluation that we used in lieu of this ideal form of evaluation.

4.1 Experimental Setup

We performed two types of evaluation, the first was a user-based annotation experiment intended to test various parameters used in the generation of the feature ranking and clustering visualisations. This was not so much an evaluation of the utility of the visualisations but rather an attempt to determine the optimal parameters for generating visualisations. The second type of evaluation represented an attempt to gauge the utility of the visualisations by simulating contexts from the grammar development process in which these visualisations are intended for use.

4.1.1 User Annotations

We performed the user annotation experiment by creating a web-interface which presented users with different visualisations of the one grammar change. These visualisations were generated using a range of different parameters, with only one parameter being tested per annotation. Users were provided with a short description of the change made to the grammar and asked to select the visualisation which they felt best captured the nature of the change.
The parameters that were varied between different visualisation were the weighting function, the number of analyses from each item used to create the feature vectors, and whether the visualisation was (1) feature ranking or (2) clustering based. For the feature weighting function, the different options were (1) the naïve item frequency, (2) the change in IDF given in Equation 3.2 and (3) the change in modified IDF given in Equation 3.4. In the case of the number of analyses used, the values tested were (1) including only the top analysis and (2) including the top ten analyses. Figure 4.1 shows an example annotation presented as a part of the experiment.

The data that we used for these experiments came from the Jacy Grammar, which was selected largely because of its healthy use of revision control. Commits were frequently made to the repository by grammar engineers, which meant that we had access to a range of snapshots of the grammar, each representing relatively discrete changes. These changes were also well documented in the changelog, which provided us with a summary of each change, which could be used as the description to be presented to the user as a part of the annotation. We selected three changes from the repository (pairs of before and after grammars) that were used for the creation of the visualisations. These changes were selected such that the modifications they
Chapter 4: Evaluation

contained were of a linguistic nature\(^1\) and resulted in a reasonable sized change in the number of items either parsing or failing to parse after the change. Participants in this study were required to either have experience developing Jacy or a more general familiarity with DELPH-IN grammars as well being able to read Japanese.

4.1.2 Grammar Breakages

This form of evaluation was intended to simulate the use of the visualisations during the grammar development process. As mentioned in Chapter 2, when seeking feedback on the effects of a change made to a grammar, grammar engineers want to know whether the change they made had the desired effect and whether it introduced any problems. We chose to focus on the latter of these two desiderata in this experiment. We performed these experiments using grammars with modifications known to have resulted in problems that were non-trivial to isolate. Visualisations of these changes were created and presented to participants who were provided with the task of playing the role of the grammar engineer. Participants were asked to hypothesise about the nature of the problem that resulted from the change, based upon a short description of what the change was intended to do. They were also asked to describe the reasoning process they used to come to these conclusions, so as to provide us with an indication of how the visualisations were being used.

Following on from the feedback from the user annotation experiment, which is outlined in Section 4.2.1, we also provided participants with extra information about the features used to generate the visualisations, in order to provide a clearer picture of why different features received the weighting they did. This information was presented as an expanded feature ranking, which while still ranked by feature weightings, also included the change in the number of items parsing with the feature in its derivation tree. An example of this expanded feature ranking is given in Table 4.1 for the same change to Jacy used to generate the visualisations in Section 3.

We used three separate changes to the ERG made during its development which were ultimately not incorporated into the grammar due to problems discovered with each change. Each change involved a different type of modification to the grammar and also resulted in a problem that was not apparent at the grammar profiling stage, but required the analysis of treebanked items before being identified. This is precisely the situation where improved feedback earlier on could assist the grammar engineer with identifying problems, thus avoiding the time consuming process of analysing changes in treebanked items.

As an example of the kinds of changes that were presented, one in particular involved an attempt to reduce the number of analyses being generated for the noun phrase *three thirty*. The issue was that one of the analyses saw it being incorrectly

---

\(^1\)Changes made to precision grammars often involve modifications to components of the system other than the grammar itself, such as the parser and preprocessing modules.
Table 4.1: Expanded feature ranking for parse $\rightarrow$ no parse category from a change made to Jacy. The feature ranking has been expanded to also include the frequency of features occurring it items pre-change and post-change to the grammar.

analysed as a determiner followed by a noun. A fix for this problem was attempted by the modification of the type `reg_or_temp_nom_rel` which then resulted in valid noun phrases containing numeric determiners such as `two days`, failing to parse. Given that we are hoping to identify the effects of this change rather than the change itself, ideally the participants would be able to flag the problem as having something to do with a component of the grammar that is involved in handling numeric determiners.

4.2 Results and Discussion

4.2.1 User Annotations

The main result that arose from the user annotation experiments was that participants found the distinctions between visualisations that they were asked to make either did not make enough sense given the context they had, or that they were not confident in many of their annotations. Part of the difficulty encountered was caused by asking the participants, none of whom were currently working on the grammar, to comment on a change that they most likely knew nothing about. When in reality, the grammar engineer is fully equipped with the knowledge of what they were intending to accomplish with a change, and exactly which components of the grammar they modified to do this. These difficulties meant that we were unable to draw quantitative results from this experiment. However, we were able to make some tentative observations from the small amount of data that we did obtain as well as from anecdotal evidence from the participants.

One particular problem that emerged, was that participants had trouble selecting a preferred visualisation when the parameter being varied was the weighting func-
tion. It turned out that the problem was in selecting between the two different 
IDF-based weighting functions. Participants preferred visualisations generated with 
either of these weighting functions over visualisations generated with the frequency-
based weighting function, but saw no difference between the two IDF-based weighting 
functions. This squares with our hypothesis that a frequency-based weighting func-
tion would not perform as well as a weighting function that accounts for the 
bias associated with frequently occurring features. It is perhaps also unsurprising 
that participants had trouble selecting between the IDF-based weighting functions. 
The modified IDF weighting function would only have resulted in significant changes 
amongst the relative weighting of features where the number of items they were found 
in did not change between grammar versions. Looking closely at the feature ranking 
visualisations reveals that for the grammar changes we used, this only occurred rarely 
within the top ten features used for creating the ranking visualisations. This may 
suggest that the modified IDF weighting function is only useful in only a handful of 
cases, or perhaps if the ranking visualisations included a greater number of features.

Another observation was that the clustering-based visualisations were often felt to 
be more useful than the ranking-based visualisations. This was reported to be because 
the clusterings simply provided more data than the feature rankings. This makes 
sense, as the clustering visualisations contain much of the information found in the 
feature ranking visualisations within the individual feature rankings of the exemplar 
items. No observations, however, were able to be made regarding the number of 
analyses which produced the preferred visualisations. It should be emphasised that 
these are only tentative observations, and require further investigation, using a refined 
evaluation methodology based on the results of this experiment.

The problems reported by the participants also provided important feedback at a 
more general level. In addition to the problems involved with exposing participants 
to an unfamiliar change to the grammar with minimal context, it also seemed that 
a major requirement for the utilisation of these visualisations was a deeper under-
standing of how they were generated and how they are intended to be used. While 
the participants in this study were presented with a brief description of how the vi-
sualisations were generated, this was insufficient for understanding how to interpret 
the visualisation. In order for tools based on these visualisation to be incorporated 
into the grammar engineering process, some training in the use of the tool would be 
required first.

4.2.2 Grammar Breakages

The results of the grammar breakage experiments were more successful than the 
user annotations. Participants were often — but not always — able to identify the 
modified components of the grammar that had caused the problem. In the case of the 
change trying to eliminate ambiguity in the phrase three thirty, participants were able 
to identify that the problem was centred around numeric determiners. Encouragingly,
both the clustering and ranking visualisations clearly indicated that a likely contender for this problem was the type `num_det_c`.

The descriptions of the reasoning processes used by participants, also revealed a number things. One of which was that, in line with the results of the user annotations, the clustering visualisations were considered to be particularly useful. It was less so the exemplar items of the clusters that were cited as being helpful though, but rather the features contained within the item, as well as the cohesion and overlap values provided for them. The features were used to determine the nature of the change being represented by the cluster, and then the cohesion and overlap features were used to determine whether this was actually an informative cluster or not. Where the exemplar items were useful was in determining the role of the features found in that item, which would most likely be less important when this task is being undertaken by a grammar engineer familiar with the function of most of the types in a grammar.

It also turned out that the expanded feature ranking information was an important source of data. In particular, the change in frequency of the features in parsing items allowed for a greater understanding of why a feature had received the weighting it had. This was particularly valuable in interpreting the clustering visualisations which included clusters with very few items in them. Due to their small size, these clusters may appear to be of limited information. In the experiments, however, some of these clusters appeared in the `parse → no parse` category. A small number of items coming from this category does not necessarily correspond to a small change in the features appearing in that category, however, since the change could have caused the number of parsing items containing that feature dropping from a large number to very few; this significant change is the very cause for there being only a few items in this category.

The utility of the expanded feature ranking lends support to the conclusion that the weighting function is an insufficient indicator of the change in a feature. While the weighting of a feature does provide a good indication of how important it is to the change, on its own it is too opaque, requiring further information such as changes in frequency to illuminate the underlying change being the feature. A beneficial enhancement of both the clustering and original feature ranking visualisations would then be to combine this type of information with the existing data.

In cases where participants had difficulty isolating the nature of the problem, a common hindrance was identified as the appearance of prominent features in the visualisation that did not correspond to any of the changes being investigated. In a small number of cases, the source of this noise could potentially have been the initial state of the grammar not being recreated perfectly, which would have introduced differences between grammar versions that did not pertain to the change being investigated. In a large number of cases, the source of the noise was clearer however. These noisy features resulted from items that had failed due to parser resources limitations. Failures due to parsing timeouts and memory limitations often occur for longer items which generate many possible analyses. The complicating issue is that these these items can be corner cases, sometimes parsing and sometimes failing, depending on variables
such as the load on the machine and other external factors. These fluctuations in items from one parse to the next, would clearly result in changes between versions of the grammar which do not correspond to the modifications made to the grammar. Since resource-dependant fluctuations are an unavoidable reality of precision grammars, this problem is one that will need to be managed. One possible solution for this would be to isolate problematic items from the creation of the visualisations by providing the grammar engineer with an option to exclude items of longer lengths.

What also emerged from the results was that participants who were more familiar with the way the visualisations were created, were better able to filter out the extraneous features and focus only on those that were pertinent to the change being investigated. This again highlights the need for comprehensive training to be first undertaken before using these visualisations.
Chapter 5

Conclusion

5.1 Concluding Remarks

In this thesis we investigated techniques for providing grammar engineers with better feedback on the effects of changes made to precision grammars. As described in Chapter 2, the need for this feedback arises from the complexity of precision grammars and the degree of analysis that is required to make modifications to them. Existing tools that assist with this process tend to provide either immediate but very high-level feedback, or very low-level incremental feedback. Furthermore, most of these tools don’t explicitly focus on the change between versions of the grammar, requiring the grammar engineer to isolate changes in the results manually. We identified the need for a tool which gives feedback explicitly on the changes made to the grammar, as well as providing a better picture of the impact of the change earlier on. Such a tool could potentially reduce the amount of time taken to identify problems with changes made to the grammar.

Our methodology, as described in Chapter 3, was to compare parser output from two different states of a grammar and then use this to create visualisations capturing the nature of the change. We investigated two distinct forms of visualisation. The first involved ordered lists of components in the grammar deemed to have been affected by the change. The other, was a set of input items to the parser intended to stand as representative examples of groups of like-effects had upon the grammar.

Through the evaluation outlined in Chapter 4, we learnt that together, both these visualisations show the potential for enhanced feedback in the grammar engineering process. In particular, it was possible to use them to locate the changes made to grammars which resulted in problems. While these were encouraging results, there were a number of problems identified with the visualisations. These included difficulties involved in interpreting the results as well as noise appearing in the results. Additionally, a more substantial form of evaluation is needed. The final section of this thesis outlines some potential avenues of improvement as well as methods of evaluation that could be pursued in future work.
We also received positive comments from the grammar engineers who helped us with this investigation, indicating that they would be interested in using a tool that provided these kinds of visualisations in the context of developing grammars they are actively working on. Something else that was clear from working with the grammar engineers, is that even after any improvements are made to address the issues discovered, a requirement for the successful use of these types of visualisations will be comprehensive training in how they are constructed and intended to be used. This is perhaps unsurprising given the complexity of the grammar engineering task; it would be rare that a new tool could be dropped into the grammar engineering process without any training. Furthermore, interaction with the people who will be benefiting from the visualisations will ultimately help refine the tool through feedback. Indeed, any of the further work presented in the next section should ideally be undertaken in the context of a dialogue with grammar engineers.

5.2 Future Work

The most needed future work is substantial road testing of these techniques in the development of an existing grammar. Only through this style of hands-on evaluation can the value of these visualisations truly be established. This could simply involve a team of grammar engineers working on a grammar for an extended period of time using the visualisations in their grammar engineering process and reporting on any ways in which this affected development. This investigation could also provide a more substantial analysis of the optimum values of the parameters used to generate the visualisations. That the grammar engineer is so firmly tied to the evaluation of these techniques speaks, again, towards the need for tools such as these to be developed in consultation with grammar engineers.

In this thesis we did not generate visualisations from items whose parsability did not change – that is, items from the category \( parse \rightarrow parse \). This category may seem uninformative, however in the case of items whose number of analyses have changed, they provide an important source of information regarding the effects of a change made to the grammar. This is particularly true in cases where any change in the number of analyses being generated is most likely undesirable, such as a refactoring change, involving a re-organisation of the grammar to provide the same analysis using a different implementation. Since the work in this thesis was of an exploratory nature, this category was excluded to avoid the complexity that would have been introduced by attempting to compare items with two corresponding sets of parse results against items with only one set. A simple way of handling this problem could be to just provide two sets of visualisations for this category, one corresponding to the parse results from the initial state of the grammar and one corresponding to the modified version of the grammar.

One of the things we learned from the evaluation was that increased information in
the form of the change in feature frequency improved the utility of the visualisations. There are certainly further statistics, readily derivable from the techniques employed in the creation of the visualisations which could also be added. Additionally, the grammar engineer’s needs may also change depending on the context. It could thus be desirable to provide the grammar engineer with some degree of control over the information that is presented, allowing them to adjust the visualisations according the context of a particular change being investigated against a particular corpus. A good example of this would be the ability to filter out items of certain lengths from the results in order to avoid the type of noise described in Chapter 4 caused by parser resource limitations. Working with grammar engineers to ascertain which additional data points are useful and how they could be best incorporated into the visualisations would be an important line of future work.

Another direction that this work could be taken in is the use of different sources of data for generating the visualisation. In addition to the full derivation trees, which represent syntactic structure, DELPH-IN grammars also produce data structures using the Minimal Recursion Semantics (Copestake et al. 2005), which represent semantic structure. Generating visualisations based on changes in these structures would likely be of use to grammar engineers since these are deep changes affecting the way the meaning of items is being analysed. Many changes made to precision grammars are intended only to affect syntactic analysis, in which case any feedback regarding changes in semantic analysis would be valuable.

In generating the visualisations, we made use of the parser output generated for successfully parsed items. Further data could be gathered by using the data structures built up for partially parsed items. One way this could be done is by using the contents of the parse chart, the data structure often used by parsers to store partial analyses during the course of parsing. Mining the contents of the parse chart has been used by Zhang et al. (2010) for the purpose of lexical acquisition; their method could be adapted to be used for the purpose of creating visualisations. Another way of including partial parse information could be to use the data structures generated during the execution of the quick check optimisation (Kiefer et al. 1999), which is often used in the parsing of precision grammars.
Bibliography


the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics (COLING 2002), 1–7, Taipei, Taiwan.


Annual Conference of the North American Chapter of the Association for Computational Linguistics, 10–18, Los Angeles, USA.


Author/s:
Letcher, Ned

Title:
Visualising the impact of changes to precision grammars

Date:
2010

Citation:

Publication Status:
Published

Persistent Link:
http://hdl.handle.net/11343/35682

File Description:
Visualising the impact of changes to precision grammars

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.