Keywords Precision Grammar, Grammar Engineering, Grammar Diagnostics,
Deep Parsing, Parse Mining

Letcher, Ned, Rebecca Dridan and Timothy Baldwin (2015) gDelta: A Missing Link in the Grammar Engineering Toolchain,
Language Resources and Evaluation.

gDelta: A Missing Link in the Grammar Engineering Toolchain

Ned Letcher · Rebecca Dridan · Timothy Baldwin

July 4, 2014

Abstract The development of precision grammars is an inherently resource-intensive process; their complexity means that changes made to one area of a grammar often introduce unexpected flow-on effects elsewhere in the grammar which may only be discovered after some time has been invested in updating numerous test suite items. In this paper, we present the browser-based gDelta tool, which aims to provide grammar engineers with more immediate feedback on the impact of changes made to a grammar by comparing parser output from two different grammar versions. We describe an attribute weighting algorithm for highlighting components of the grammar that have been strongly impacted by a modification to the grammar, as well as a technique for clustering test suite items whose parsability has changed, in order to locate related groups of effects. These two techniques are used to present the grammar engineer with different views on the grammar to inform them of different aspects of change in a data-driven manner.

1 Introduction

In this paper we investigate techniques for generating feedback on the impact of changes made to precision grammars, and present the gDelta tool, which is intended to combat a significant drawback of linguistically accurate hand-crafted grammars — the considerable amount of time and resources involved in their development. gDelta is intended for use during the grammar development cycle,

Ned Letcher
Department of Computing and Information Systems, The University of Melbourne, Victoria 3010, Australia
E-mail: ned@nedletcher.net

Rebecca Dridan
Department of Informatics, University of Oslo, Oslo, Norway
E-mail: rebecca@dridan.com

Timothy Baldwin
Department of Computing and Information Systems, The University of Melbourne, Victoria 3010, Australia
E-mail: tb@ldwin.net

to be presented to grammar engineers as immediate feedback on the impact of a change or set of changes made to a grammar. This feedback is intended to assist grammar engineers in more readily understanding how the grammar has been affected, thus reducing the amount of time required to track down issues introduced by changes to the grammar.

Previous work that aims to provide feedback for use in the grammar engineering process (Oepen and Carroll 2000; van Noord 2004; Waterman 2009) has focused on the analysis of parser output from just one version of a grammar. Our approach differs in that it highlights the differences between two versions of a grammar: one from before and one from after a change made to a grammar. This work can be conceptualised as the development of a `diff` tool that operates over parser output from two different versions of a grammar run across the same corpus. Rather than trying to improve upon or replace existing development tools, we see this tool as being complementary, filling a gap in the grammar engineering toolchain. gDelta was developed specifically for use with grammars developed within the DELPH-IN community, [1] however the underlying techniques that drive gDelta and are described in this article are largely framework agnostic, and could be readily applied to other grammar engineering frameworks. Additionally, these techniques work cross-linguistically, as demonstrated by the application of gDelta to grammars of English and Japanese.

The remainder of this paper is structured as follows. Section 2 briefly expands upon the concept of a precision grammar and outlines the grammar engineering process, before presenting an overview of existing tools available to grammar engineers. Section 3 outlines our methodology, including the resources we used and the underlying techniques that gDelta makes use of. Section 4 provides a walk-through of the output that gDelta generates and outlines how it is intended to be used. Section 5 outlines a small-scale evaluation of the techniques introduced in this paper and contains a discussion of the results. Section 6 provides a brief overview of the implementation details of gDelta, along with the requirements for running gDelta and how to access the source code. Section 7 identifies further work to be pursued in his area, and Section 8 concludes the paper.

2 Background

2.1 Grammar Engineering

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 paper is Head-driven Phrase Structure Grammar (HPSG: Pollard and Sag 1994). Examples of other grammar engineering approaches include the CoreGram project (Müller 2013), also based on HPSG; the ParGram project (Butt et al, 1999; Butt et al, 2002) based on Lexical Functional Grammar (Dalrymple 2001); the eXtensible MetaGrammar tool (Crabbé et al 2013), which can be used to create tree-based grammars such as Tree Adjoining Grammar (Joshi and Schabes 1997) and Interaction Grammar (Guillaume and Perrier 2009); and also DotCCG (Baldridge

[1] `http://www.delph-in.net`

et al 2007), a tool used to develop Combinatory Categorial Grammars (Steedman 2000).

These foundations in formal syntax mean that precision grammars all share the property that they draw a sharp distinction between sentences that are grammatical and those which are not. Compared with induced grammars — as are used in most treebank parsers — precision grammars tend to generate far more detailed analyses and are able to capture much more complex linguistic constructions (Bender et al 2011). Precision grammars are also usually able to capture 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*.

Grammar engineering is the process of developing and extending the coverage of precision grammars as well as improving the quality of existing analyses. It is a resource-intensive process, requiring grammar engineers with both a linguistic background as well as an understanding of the formalism in which the grammar has been implemented. A number of factors combine to make grammar engineering a slow and time-consuming process. One is the considerable size and complexity of these grammars, with the various components involving a high degree of interaction between each other, meaning that modifications made to one part of the grammar can often produce unexpected flow-on effects in other areas of the grammar. Another 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 important that the validity of analyses produced by the grammar does not degrade, and that the grammar does not stop being able to parse 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 analyses produced by the grammar is referred to as *grammar profiling*. This involves the maintenance of test suites which facilitate the monitoring of various signals, providing a means of ensuring that the grammar does not regress. In the DELPH-IN community, test suites are maintained through the use of *profiles*, which contain sets of *items* and information pertaining to how the grammar performed over the items. An item consists of a string from the target language, which, depending on the purpose of the test suite, can be naturally occurring or constructed to illustrate a particular phenomenon, and can also be marked as grammatical or ungrammatical. One of the aims of the TSNLP project (Oepen et al 1997), which the DELPH-IN profiling technology originates from, was to additionally annotate items with grammatical phenomena they exemplified as a means of further evaluating and debugging the performance of grammars. The complexities involved in creating and maintaining such phenomenon annotations have, however, meant that this practice has not been adopted.

The most prominent aspect of the grammar that must be monitored is the range of linguistic phenomena and lexical items that it is able to handle — the coverage of the grammar. This is monitored by test suites constructed to track the extent to which a grammar *undergenerates*. Test suites can also contain ungrammatical or negative items used to test the degree to with a grammar *overgenerates*. In addition to these signals, the number of analyses per item is also monitored. While it is true that ambiguity is an inherent feature of language that cannot be eliminated, it is important to remove spurious ambiguity where possible. Gram-

mar profiling tools also provide information regarding changes in the operational performance of the grammar, such as the time taken to process test suites and the number of operations required for parsing. Just as with testing in traditional software development, unexpected changes in any of these aforementioned statistics are potential symptoms of introduced errors. For further discussion of the requirements of grammar profiling tools, see Oepen and Flickinger (1998).

A limitation of grammar profiling tools is that they are not able to detect the misanalysis of items, i.e. when the analysis of an item does not match its structure. This is of particular concern when developing precision grammars as their architecture tends to be complex, reflecting the inherent complexity of language, and just as linguistic phenomena interact with each other, so too do the implementations of their analyses. This means that a modification to a grammar pertaining to a particular phenomenon can often result in unanticipated flow-on effects in other areas of the grammar.[2]

A resource that can be used to monitor the accuracy of analyses is the *treebank*, a corpus annotated by selecting a good analysis from the alternatives produced by the parser. Whenever a change is made to the grammar that alters the stored analysis of a treebanked item, the analysis provided by the previous version of the grammar can be used to verify that the change has not led to a degradation in quality. When using treebanks for this purpose, a necessary step in the grammar engineering cycle is the updating of items affected by the change to the grammar. This is performed to catch errors resulting from the change, and also to periodically synchronise the treebank to the current state of the grammar. In the DELPH-IN grammar engineering pipeline, this dynamic treebanking methodology (Oepen et al 2002) takes advantage of previously recorded annotations by automatically applying (where possible) the outcomes of decisions regarding the resolution of local ambiguity. This greatly reduces the number of decisions an annotator must make following a change, however it remains the case that following a significant change to a grammar, the treebanking stage can be a potentially labour intensive process.

2.2 A Gap in the Toolchain

Grammar profiling and treebanking are both vital to the grammar engineering process, each representing opposite extremes in the level of feedback they provide: grammar profiling providing immediate but course-grained feedback on the coverage and performance of the grammar, and treebanking providing slower, fine-grained feedback on an item-by-item basis. The grammar profiling stage can locate glaring problems such as coverage levels suddenly dropping or not changing when expected to, but the detailed level of feedback provided by treebanks is crucial to catching unanticipated problems not visible in the grammar profiling stage, as well as more generally building up a picture of how the change impacted the grammar. Unfortunately, the inspection and updating of treebanks can be a slow process,

[2] In the DELPH-IN grammar engineering formalism, rules and lexical entries can inherit constraints from multiple supertypes. This allows for commonly occurring constraints to be collected together, which is desirable both from the point of view of capturing linguistic abstractions and also avoiding repetition, but at the cost of increasing the chances of surprises.

and an all too real possibility for the grammar engineer is locating a serious problem after considerable time has been invested in updating the treebank. When this occurs, the grammar engineer must resolve the problem in the grammar and start the process over again.

We argue that it would be desirable for there to be a tool which offers a middle ground between these two levels of feedback, one which makes the impact of the change more transparent earlier on in the grammar engineering cycle, helping the grammar engineer more quickly gain a superficial understanding of the impact of the change over the test suites. In the next section, we outline a range of existing tools and approaches that grammar engineers could potentially use to gauge the impact of a change made to a grammar. Each approach is assessed on the basis of how well it can be used to provide the desired kind of feedback.

2.3 Previous Work

van Noord (2004) proposed a *grammar error mining* technique for detecting errors in grammars. This involves assigning a parsability score to each word n-gram in a corpus, based on the ratio of the number of times each n-gram occurs in successfully parsed items over the number of items it occurs in the whole corpus. The end result is a list of n-grams, ordered by parsability, with the lowest scoring n-grams representing those most likely to be indicative of problems in the grammar. This technique has been further improved by Sagot and de La Clergerie (2006) and de Kok et al (2009). An advantage of this error mining approach is that the only input required is a binary *success* or *failure* value for each item, meaning that it is agnostic towards the implementation of the parser and grammar. A drawback to this approach, however, is that the results are presented in terms of surface string n-grams, whereas information pertaining to the internal components of the grammar would be more informative to the grammar engineer. This is indicative of the larger problem, which is that without an analysis from the grammar, the error mining input is limited to unstructured text.

Two error mining approaches which do utilise structured input to yield further insight are that of Goodman and Bond (2009) and Gardent and Narayan (2012). Goodman and Bond (2009) introduce Egad, a tool which leverages the ability of precision grammars to both generate and parse in order to identify errors involved in generation. For each successful parse, they use the corresponding semantic form to generate the set of possible paraphrases, flagging a sentence as a failure if the original surface form cannot be generated. They then use a maximum entropy classifier to find n-grams over paths in the derivation tree which are predictive of generation failure, resulting in a better picture of where the underlying error in the grammar might lie. Similarly, Gardent and Narayan (2012) extend the error mining approach of de Kok et al (2009) to develop a suspicion measure for dependency subtrees, applying this to input trees for which they are unable to generate a surface form.

These automated approaches towards error detection in grammars are useful for directing the development of a grammar — they are good at locating gaps in parsing and generation coverage — however they are ill-suited to providing feedback on immediate changes made to a grammar, as there is no guarantee that the problems they report will be related to changes just made to the grammar.

This is due to the use of a single version of the grammar in the input: without comparison to a previous version, there is no bound on how far back the errors could have been introduced, nor are these introduced errors distinguished from known gaps in the grammar.

The tool used for grammar profiling and treebanking in the DELPH-IN community is a software package called [incr tsdb()] (Oepen and Carroll 2000). It facilitates the construction of test suites designed to measure overgeneration and undergeneration in the grammar, as well as changes in the number of analyses produced per item. [incr tsdb()] is able to be used in a comparative mode, comparing the coverage and performance of two different versions of a grammar. This makes it a valuable tool for gaining immediate feedback on the impact of a change just made to a grammar. However, it suffers from the previously-mentioned limitations of grammar profiling tools, i.e. that this information offers a coarse-grained level of feedback on the impact of the changes made.

A tool which can be used to shed more light on changes in terms of the underlying properties of the grammar, is Oceanography (Waterman 2009) from the XLE grammar development environment (Crouch et al 2014). Oceanography works by processing parser output and collating statistics such as the counts of constructions and properties and the context in which they occurred. Dost and King (2009) describe making use of Oceanography to gauge the impact of modifications made to a grammar by comparing the relative frequencies of the modified constructions between the initial version of the grammar and the modified version. Unexpected changes in frequencies of constructions are deemed to be symptomatic of potential problems introduced by the change. This comparative use of Oceanography between two different versions of a grammar comes the closest to meeting the desired objectives.

2.4 Filling the Gap

In order to try to fill this gap in the grammar engineering toolchain, we have developed gDelta, which, like the parse-result mining of Oceanography, involves processing output produced by the parser. Our approach is similar to the comparative use of Oceanography in Dost and King (2009), but improves upon this approach, firstly through the use of a single invocation and unified output, removing the need for manual comparison of two separate runs over two grammar versions, and secondly by performing analytics over the combined results in order to highlight significant changes between the parse results.[3] Unlike grammar error-mining techniques, gDelta does not assign judgements of suspicion to results, but instead attempts to provide a sufficient amount of diagnostics on the impact of the changes between the two grammar versions, from which the grammar engineer can draw their own conclusions.

Rather than trying to improve or replace any of the existing tools and approaches mentioned above, we see gDelta as being complementary, representing a new addition to the grammar engineer's toolchain with a distinct use-case from the existing tools. More concretely, this use-case is to provide an intermediate level

[3] Note that it is not possible to perform direct comparative evaluation of gDelta and Oceanography, as they have been developed to work with different grammar engineering frameworks (the DELPH-IN stack and XLE, respectively).

of feedback, allowing grammar engineers to identify unexpected and potentially detrimental consequences of their change before commencing manual inspection of changed test items.

3 Methodology

For the development of gDelta, we used the Japanese grammar Jacy (Siegel 2000) and the LinGO English Resource Grammar (ERG: Flickinger (2002)), both HPSG-based precision grammars developed within the DELPH-IN community. Instead of creating our own changes to these grammars from scratch, we used real changes made in the course of the development of these grammars. This ensured we would be working with sample changes of the kind that would occur during the grammar engineering process. For our test corpus, we selected test suites used in the development of the grammars. In the case of Jacy, this was the Tanaka Corpus (Tanaka 2001) and for the ERG, this was a selection of different corpora found in the LinGO Redwoods treebank (Oepen et al 2002) as well as the WeScience treebank (Ytrestøl et al 2009).

3.1 Preliminary Steps

For each change to a grammar being investigated, we began by taking two versions of the grammar: the initial version of the grammar (G) and the modified version (G'). These were then used to parse the same corpus using the PET run-time parser (Callmeier 2002). In order to keep the run-time of the parser within reasonable limits (with an eye to real-time feedback in live grammar engineering environments), we restricted the analyses the parser returned for each item to the 10 best according to the parse selection model. Limiting the number of analyses to be found (while still being able to determine the total number of analyses in the parse forest) improves the parsing speed due to the selective unpacking (Zhang et al 2007) used by PET.

For each grammar pair (G, G') we then created three sets of parse results: (1) parse results for items which parse with G but not G' ($parse \rightarrow no\ parse$); (2) parse results for items which do not parse with G but do with G' ($no\ parse \rightarrow parse$); and (3) parse results for items that parse under both versions of the grammar but receive different numbers of analyses in their parse forests ($parse \rightarrow parse$). In the latter case, which contains analyses from both versions of the grammar, we subdivided this set into a further two: (1) analyses from G ($\boldsymbol{parse} \rightarrow parse$); and (2) analyses from G' ($parse \rightarrow \boldsymbol{parse}$). These four categories are intended to reveal different kinds of effects resulting from changes to the grammar and are the basis of much of gDelta's output.

3.2 Attribute Extraction

As was argued in Section 2, in order to gain the kind of feedback the grammar engineer is after, it is necessary to leverage more information than just the parsability of an item. Just as Oceanography does, we look for further information in the

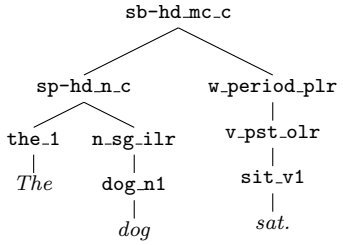

Fig. 1 Derivation tree produced by the ERG for the sentence *The dog sat.*

```
dog_n1 := n_-_c_le &
 [ ORTH < "dog" >,
   SYNSEM [ LKEYS.KEYREL.PRED "_dog_n_1_rel",
   PHON.ONSET con ] ].
```

Fig. 2 The lexical entry for the word *dog* in the ERG

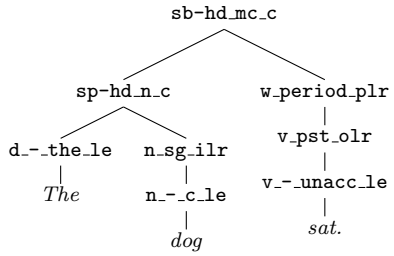

Fig. 3 The derivation tree from Figure 1 with lexical entries replaced by their lexical types

output 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 derivation trees generated by the parser, which represent the syntactic analysis of successfully parsed items.

An example derivation tree for an English phrase parsed with the ERG can be seen in Figure 1. In this example, just the labels of the nodes are shown, which are — with the exception of the terminals and preterminals — the identifiers of the rules used to derive the subtree at each node.[4] In reality, each node stands for an elaborated attribute value matrix, containing all the constraints involved in building up the derivation for that subtree.

We extracted the rule identifiers from nodes of the derivation trees to be used as the attributes for use in gDelta. In the case of the leaves of the tree, extracting the surface form of the word would have led to data sparsity problems due to the infrequent occurrence of most words. This would also likely have occurred had we used the preterminal nodes, which correspond to the lexical entry for that word.

[4] An interactive demo of the ERG can be found at `http://erg.delph-in.net/logon`, where rule identifiers can be viewed by hovering over nodes of parse trees from successfully parsed sentences.

Instead, we extracted the lexical type of each lexical entry, requiring a simple lexicon lookup. In accordance with the highly lexicalised nature of HPSG, it is a convention of DELPH-IN grammars that lexical entries inherit most of their constraints from a lexical type, and contain little information themselves. Figure 2 shows the lexical entry for the word *dog* in the ERG. It is composed of a lexical type, `n_-c_le`, which abstracts the constraints common to count nouns, as well as a small amount of information specific to the word *dog*. Figure 3 shows the derivation tree from Figure 1 with lexical entries replaced by their lexical types.

In the context of HPSG, these two kinds of attributes (rule identifiers and lexical types) are associated with phrases and words — subtypes of the *sign type*, which pairs constraints on both phonological form and syntactico-semantic function. The type hierarchy, which forms the backbone of DELPH-IN grammars, also includes categories of types that abstract functionality different to that of sign types. Limiting gDelta's input to the portion of the type hierarchy which can be readily extracted from the derivation trees has the advantage of making this approach more portable across other grammatical frameworks as well as being simpler to implement since consultation of the grammar itself is not required. In Section 7 we discuss potential improvements to gDelta that involve expanding the input to more of the type hierarchy as well as other data structures produced by the parser.

A further issue that arose in the attribute extraction process was how attributes are extracted from items that yield more than one analysis, which for non-trivial input is the norm rather than the exception. One possibility would be to extract the union of attributes across all of the recorded analyses. But since the probability assigned to some of the analyses by the parse selection model could be quite low, this may in fact result in a poorer reflection of how the parse results have been affected by the change. Since it is not clear at which point analyses should be excluded from contributing their attributes, we left this as a user-definable parameter to gDelta, with the default being to use only the single best analysis.

The attribute extraction step is the one component of the gDelta methodology which is DELPH-IN/HPSG specific. In order to apply a gDelta style approach to other grammatical frameworks, a procedure for extracting appropriate attributes from derivation trees—or indeed other data structures—must be developed. The attribute ranking and clustering algorithms described in the remainder of this section are agnostic with respect to the underlying grammar framework, only requiring a list of attributes and their item counts across the parser output produced by the two versions of the grammar.

3.3 Attribute Ranking

Having extracted attributes from the derivation trees of parsed items, we then needed a means of highlighting attributes impacted by the changes made to the grammar. This was first attempted by ranking attributes by the relative change in the number of parsed items that contained the given attribute for each version of the grammar, sorted in either ascending or descending order, depending on whether decreases or increases were being investigated. A problem with this ranking was that attributes considered to be particularly relevant to the change were sometimes only found in a small number of items, meaning they were ob-

scured by changes in more commonly-occurring attributes. When developing the item clustering approach outlined in Section 3.4, we also noticed that using the change in item frequency for the weighting of attributes biased the results towards lower numbers of clusters. We combated these two problems by using an alternative attribute weighting function based on the difference in the inverse document frequency (IDF) of attributes. Since the IDF of attributes will be low for those that occur frequently in a set of parse results and high for those that occur infrequently, the difference in IDF is effectively the change in rarity of attributes between parse results.

The IDF of attribute i in grammar G is given by Equation 1. P_G is the number of parse outputs for grammar G and $p_{i,G}$ is the number of parse outputs that contain attribute i; the denominator is offset by one to avoid zero-division errors when the attribute does not occur in any of the parsed items. This weighting function for an attribute i is given by Equation 2, where G is the initial version of the grammar and G' is the new version of the grammar.

$$\mathrm{IDF_{i,G}} = \log \frac{|\mathrm{P_G}|}{1 + |\{\mathrm{p_{i,G}} : \mathrm{f_i} \in \mathrm{p_{i,G}}\}|} \tag{1}$$

$$\mathrm{W_i} = \mathrm{IDF_{i,G'}} - \mathrm{IDF_{i,G}} \tag{2}$$

An issue we discovered with this weighting function is that when the frequency of an attribute is not affected by a change, the attribute can receive a non-zero weighting due to the fact that the total number of items parsing will likely change between grammars, yielding a change in IDF. This would be confusing for the user, suggesting an impact upon some attributes where none exists. This was solved by modifying the IDF calculation to use the total number of items parsing across both versions of the grammar, which has the effect of assigning unchanged attributes a weight of zero, while preserving the relative weighting of other attributes. The modified IDF formula is given in Equation 3.

$$\mathrm{IDF_{i,G,G'}} = \log \frac{|\mathrm{P_G}| + |\mathrm{P_{G'}}|}{1 + |\{\mathrm{p_{i,G}} : \mathrm{f_i} \in \mathrm{p_{i,G}}\}|} \tag{3}$$

Armed with the item frequency and IDF-based weighting functions, rankings could then be created across the sets of attributes extracted from the four different categories of parse change. Table 1 and Table 2 show attribute rankings for the categories of *no parse* → *parse* and *parse* → *no parse* respectively, comparing the top 10 changes across both weighting functions. These rankings were generated from a change made to Jacy during the course of its development which involved extensive changes to the handling of verb phrases. The rankings in Table 1 are sorted in descending order, while those in Table 2 are sorted in ascending order, reflecting the direction of change that is of interest for each category. It should be noted that while the set of attributes for each parse change category only includes attributes found in items from that category, their changes in item frequency and IDF are calculated globally across all items, meaning that it is possible to see, for instance, attributes in the *no parse* → *parse* category with decreases in item frequency and IDF weightings.

Looking at Table 1, which shows attributes extracted from items which now parse as a result of the changes made to the grammar, we see in Table 1a that ranking by item frequency places the attribute `vn-light-rule` at the top of the

	Δ Item frequency	Attribute
1	1566	vn-light-rule
2	1363	vstem-vend-rule
3	130	adversative-trans-pass-passcmorph-end-lex
4	76	i-adj-neg-stem-lex
5	69	v-past-tense-end-minusahon-tmorph-lex
6	59	caus-trans-obj-passcmorph-end-lex
7	58	hi-adj-s-rule
8	56	caus-intrans-scope-passcmorph-end-lex
9	51	kit-lexeme-c-stem-infl-rule
10	51	head-complement2-rule

(a) Change in item frequency weighting

	Δ IDF	Attribute
1	2.424	caus-intrans-scope-passcmorph-end-lex
2	1.525	garu-sbj-change-rule
3	1.382	v8-c-minusshon-stem-lex
4	1.332	caus-trans-obj-passcmorph-end-lex
5	1.039	adversative-trans-pass-passcmorph-end-lex
6	0.667	ke-lexeme-infl-rule
7	0.603	vn-light-rule
8	0.404	comp-prpstn-lex-questarg
9	0.404	hes-lex
10	0.404	pure-aspect-progressive-shon-c2-stem-lex

(b) Change in IDF weighting

Table 1 Top 10 attribute rankings generated from items in the *no parse* → *parse* category from a change made to the JACY grammar during its development

ranking, being found in 1566 more parsing items than before the change, followed by `vstem-vend-rule`, with a change of 1363 parsing items. Both these attributes were directly altered by the grammar engineer, with the increases in other attributes being due to interactions with changes made to the grammar. Looking at the ranking generated using the IDF weighting in Table 1b, we see that using a weighting function based on the change in rarity of attributes has moved `vn-light-rule` down into the seventh position, and `vstem-vend-rule` has completely dropped out of the top 10.

Comparing the rankings produced using the two different weightings, we see that attributes occurring higher in the item frequency-based ranking tend to be pushed down in the IDF-based ranking, replaced by changes in rarer attributes. One of the effects of this emphasis on subtler, more nuanced changes found in the IDF ranking is that of pushing lexical types, whose distribution exhibits more of a long tail, higher up the ranking. We see these two different types of ranking as being complementary, with the item frequency weighting allowing the grammar engineer to build up a broad picture of how the attributes were affected, and with IDF-based weighting highlighting potentially interesting attributes from the long tail of item frequency changes. Within the final GDELTA interface, these two signals can be combined, by — for instance — using item frequency as the primary sort key and the IDF weighting as a secondary sort key.

During the development of GDELTA, we noticed that there were occasions where it was useful to collapse increases and decreases in attributes together. When viewing attribute rankings for the category *no parse* → *parse*, positive changes are

	Δ Item frequency	Attribute
1	-3037	aux-vend-rule
2	-643	head-specifier-rule
3	-121	lightverb-pass-end-lex
4	-36	vend-vend-rule
5	-28	pure-aspect-progressive-v-stem-lex
6	-26	topic-nobj-lex
7	-25	te-adjunct-lex-t
8	-24	vn-intrans-lex
9	-13	v-finite-present-plain-negative-end-lex
10	-12	rel-cl-sbj-gap-rule

(a) Change in item frequency weighting

	Δ IDF	Attribute
1	-6.329	aux-vend-rule
2	-4.642	lightverb-pass-end-lex
3	-2.383	vn-light-obj-change-rule-noun
4	-0.404	frg-pp-int
5	-0.404	pron-thirdsgneut-ref-lex
6	-0.179	light-shon-c-stem-lex
7	-0.124	aspect-c-stem-lex
8	-0.102	infinitive-lexeme-v-infl-rule
9	-0.079	perspective-arg1emp-shon-c-stem-lex
10	-0.067	adv-p-lex-np-nonexh

(b) Change in IDF weighting

Table 2 Top 10 attribute rankings generated from items in the *parse* → *no parse* category from a change made to the Jacy grammar during its development

of particular interest, and conversely, negative changes are more interesting in the case of the *parse* → *no parse* category. But in the case of the remaining two parse change categories, which contain items with changes in the number of parses they received, both positive and negative changes are potentially relevant. Rather than introduce an additional signal via another weighting function, we opted to modify the existing IDF weighting function to be the absolute magnitude of the change. This version of the IDF weighting function (as given in Equation 4) is the one used in gDelta[5] and referred to throughout the remainder of this paper.

$$\mathrm{W_i} = |\mathrm{IDF_{i,G',G}} - \mathrm{IDF_{i,G,G'}}| \tag{4}$$

3.4 Item Clustering

Information on differences in the correlation between parsability and individual types is a valuable aid in gisting how the grammar has changed. However, the interaction between attributes impacted by the grammar change is often subtle, and only truly understood relative to different combinations of attributes and the corresponding items. As a second diagnostic facet of gDelta, we therefore cluster items based on parsability data and the attributes present in the associated derivation tree, and present the items associated with each cluster. As with the attribute

[5] Since it is not clear that this will always be the most useful means of presentation, we also made this a system parameter, which allowed for the selection of the signed version of the IDF function — rather than the unsigned default — to be used throughout gDelta.

rankings, the clustering is performed separately across the different categories of parse change.

For the underlying clustering algorithm, we chose k-means clustering for its simplicity and run-time efficiency. k-means clustering is an iterative algorithm which partitions a set of observations into k clusters. In each iteration, every observation is assigned to the cluster with the closest mean, and the mean of each cluster is subsequently updated. This process of assign-and-update is repeated until the clusters converge and the assignment of observations no longer changes. We selected Euclidean distance as the metric for calculating distances between points. To generate the item clusters, each item is first converted into a weighted attribute vector in n-dimensional space, where n is the total number of attributes located across all parse results from both versions of the grammar.

As previously mentioned, we found that using the change in item frequency to weight the attributes had the effect of strongly biasing the results towards lower numbers of clusters. We suspect this was caused by large changes to frequently occurring attributes dominating other changes, resulting in item vectors lying close to each other in Euclidean space and thus not being readily separable. The IDF-based weighting function solves this problem by picking up on more nuanced changes to attributes. Item vectors were then weighted by assigning, for each attribute, the value returned by the function given in Equation 4 if the attribute occurred in the derivation tree for the item, and zero if it did not.

A decision to be made when using k-means clustering is how the centroids of the clusters are initialised. Since k-means is an expectation-maximisation algorithm, there is no guarantee that the final solution will be the global optimum. If a poor selection of initial seeds is made then the solution may be a local optimum a long way from the global optimum. We tried to reduce the likelihood of this by selecting centroids with a good spread across the entire set of observations. This was done by pseudo-randomly selecting a point to be the centroid of the first cluster, then selecting each other initial centroid to be the furthest point from the mean of the seeds 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. Since the grammar engineer does not know how many types of changes there may be in the parse results, the best value of k is unknown. This problem is commonly solved by repeatedly performing the clustering across a range of values of k and selecting the value which yields 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 is to minimise the distance from each observation to its centroid. This approach has the drawback of favouring higher numbers of clusters and requires the introduction of a penalty for each increment of k to offset this bias.

We chose to compare the quality of the clustering for different values of k using the Silhouette algorithm (Rousseeuw 1987), which provides a means of evaluating the fit of an observation within a set of clustered observations. 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 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 -1 to 1, with -1 indicating poor clustering and 1 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.

$$s(i) = \frac{b(i) - a(i)}{\max\big(a(i), b(i)\big)} \tag{5}$$

The final clustering is done by performing k-means for values of k between 2 and 6, and selecting the value which resulted in the highest Silhouette score. A maximum of 6 was selected after repeated observations across different grammars and grammar changes showed that the number of clusters selected rarely exceeded 4. The result is a set of clusters for each of the parse change categories, with each cluster intended to capture a group of related effects on the parse results. In order to characterise the nature of these effects, the closest item to the centroid of each cluster is taken as the exemplar and displayed with the cluster. In order to make more transparent the changes in attributes that the clustering has picked up on, we include in the clustering visualisations the top-ranked attributes of the exemplar items, as well as statistics intended to illustrate their relevance to the cluster. An example of the clustering output is provided in Section 4.3, along with an explanation of the additional signals provided in the clustering visualisations.

4 gDelta Walkthrough

The output produced by gDelta consists of a series of HTML pages for viewing within a web browser. All data required for displaying the output is contained within the HTML files, so it can be viewed offline and independently of the grammar versions being compared. The output includes five different pages, each intended to provide a different facet of feedback. This section contains an overview of these pages. The sample output was generated from an attempted change to the ERG intended to expand the coverage of extracted complements with a right-node-raising analysis. This change was also used as a part of the evaluation of gDelta outlined in Section 5 and is further described there.

4.1 Summary Page

The Summary page is intended to be the first port of call for the grammar engineer after running gDelta. It provides a high level overview of the impacts that the grammar changes had over the profiles. The first half of the page is shown in Figure 4 and the second half is shown in Figure 5. The first half provides an overview of various grammar profiling diagnostics, a breakdown of the number of items found in each category of parse change — as well as the number of items from these categories that could be used for extracting attributes — and a summary of the attributes with the top increases and decreases in item frequency.

The second half of the Summary page contains separate tables of attributes found in each category of parse change. Each table has columns containing the attribute's change in IDF (referred to as *weight*), its change in item frequency

Summary | Attributes | Clusters | Items | Errors

Overview

Grammar	erg
Previous version	b
New version	b3
Total items	11558
Profiles	ws01, ws02, ws03, ws04, ws05, ws06, ws07, ws08, ws09, ws10, ws11, ws12, ws13

	Previous	New	Change
Coverage	89.27%	89.26%	-0.01%
Parsing items	10318	10317	-1
Mean readings per item	7.52	7.51	-0.01
Mean readings per parsing item	8.43	8.42	-0.01
Total readings	86945	86828	-117
Types in grammar files	5250	5250	0
Distinct attributes in parse results	944	945	1
Errors	603	601	-2

Changes to parsability	Items	Used
no parse → parse	6	6
parse → no parse	7	7
parse → parse	86	86
parse → parse	86	86

Top Attribute Changes

Attribute	Increase	Attribute	Decrease
mrk-nh_nom_c	7	hd_xcmp_c	-16
np-np_crd-t_c	7	hd-hd_rnr-nv_c	-11
pp_-_i_le	6	pp-pp_crd-t_c	-8
n_pl_olr	5	flr-hd_nwh-nc_c	-8
np_frg_c	5	hd_imp_c	-7
hdn-aj_rc_c	4	mrk-nh_cl_c	-6
cl-cl_runon_c	4	cl-cl_crd-t_c	-5
n-hdn_cpd_c	4	w_comma-nf_plr	-4
np-hdn_cpd_c	4	mrk-nh_evnt_c	-4
hd-aj_cmod_c	4	np-hdn_num-cpd_c	-3
p_vp_bse_le	4	root_informal	-3
flr-hd_rel-fin_c	3	v_3s-fin_olr	-3
root_inffrag	3	cl-np_runon_c	-2
n_-_mc_le	3	aj_-_i-one-nmd_le	-2
v_pas_odlr	3	det_prt-nocmp_dlr	-2
v_prp_olr	3	root_strict	-2
n_pp_c-gr-of_le	3	aj-hd_scp_c	-2
hd-aj_vmod_c	3	hdn_np-num_c	-2
v_np_le	3	n_-_pr-it_le	-2
np-hdn_nme-cpd_c	3	v_np_be_le	-2

Fig. 4 The first half of the Summary page provides an overview of diagnostic information pertaining to the impact of changes made to the grammar

(referred to as *change*), and also the number of items from this category that the attribute occurred in. This third value is provided in order to determine the relevance of an attribute to the category, since the changes in item frequency and IDF are calculated globally across all items. Each column can be interactively sorted on, allowing the grammar engineer to explore different aspects of how the parse results were affected. Clicking on an attribute in these tables navigates to that attribute's position in the Attributes page described in the next section.

The *parse* → *no parse* ranking in Figure 5 illustrates the complementary nature of the different signals presented by the interface. When the Change column is reverse-sorted, the `hd_xcmp_nom_c` attribute rises to the top. However, the rela-

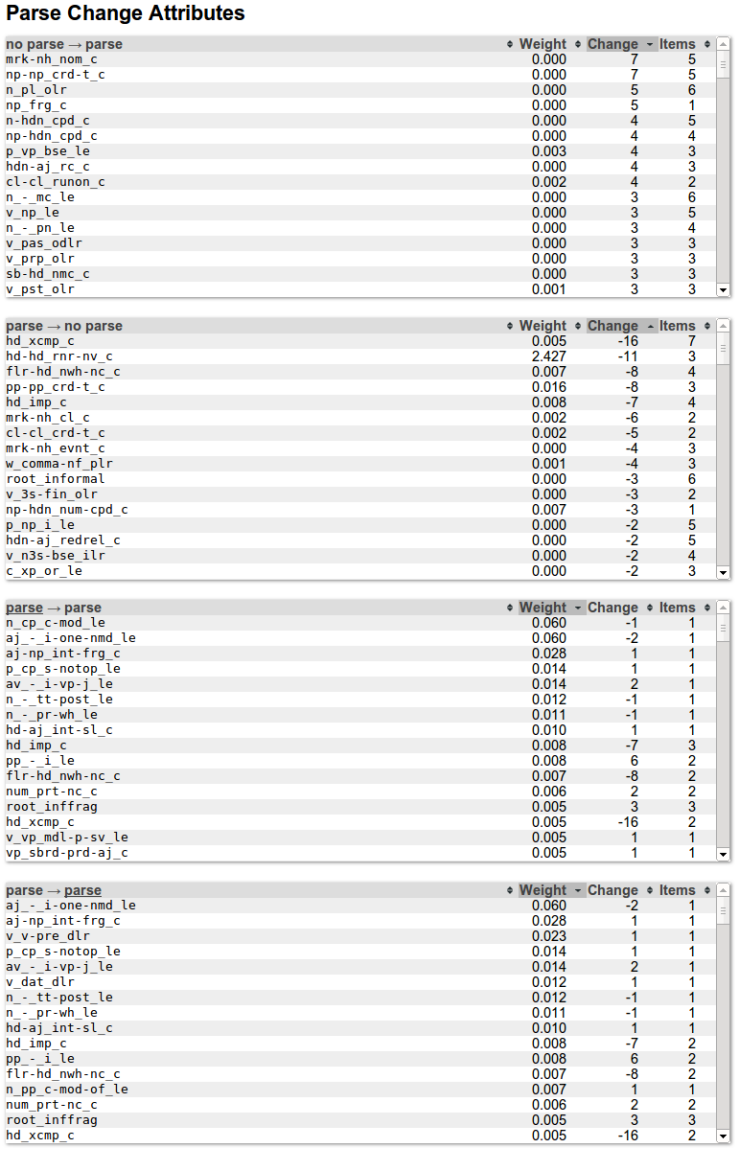

Fig. 5 The second half of the Summary page shows the top-ranked attributes across each of the different categories of parse change

tively high change in IDF of the `hd-hd_rnr-nv_c` attribute, which is involved in the right node raising analysis, correctly suggests that this attribute could be of more interest.

4.2 Attributes Page

The Attributes page aims to provide more comprehensive information pertaining to how the occurrence of attributes in the parse results was affected by the changes made to the grammar. It contains a single table listing every attribute extracted by gDelta from both versions of the grammar. The different columns of the table contain the following information for each attribute: the size of the change in the item frequency, the signed change in the item frequency, the IDF-based weighting, the number of items under the previous grammar containing the attribute, the number of items under the new grammar containing the attribute and lastly, a column indicating whether the attribute was present in the source files of only one or both versions of the grammar. This last column is intended to provide an indication of when an attribute has been added to or removed from the grammar, making the cause of an attribute's frequency changing to or from zero clearer.

All of the columns can be used as sort keys (secondary and tertiary sort keys etc. are also supported), allowing the grammar engineer to refine the feedback they receive towards a specific aspect of change that is of interest to them. There is also a text input box which can be used to filter on the name of attributes. This is useful for exploiting type-naming conventions that are commonly employed in DELPH-IN grammars. For instance, in the ERG, filtering on the substring `_le` has the effect of restricting results to lexical entries. Clicking on an attribute name in the table navigates to the Items page showing only those items which contain the attribute in the parse results.

4.3 Clusters Page

The Clusters page provides a visualisation of the clusters produced by gDelta for each of the different categories of parse change. The clusters are intended to represent different patterns of effects found in the changes between parse results. Each cluster is initially presented with only the text from the exemplar item of the cluster and the cluster's Silhouette score, which gives an indication of the fit of the items in the cluster, as described in Section 3.4. More details are available for each cluster by clicking on the Details button. This exposes a table containing the top 5 attributes found in the exemplar item as well as a table containing all items in the cluster. Figure 6 shows the two clusters for the parse change category *parse* $\rightarrow$ *no parse* with the first cluster expanded. For the exemplar attributes, in addition to presenting the attribute weighting, the cohesion and overlap of each attribute is also given. The cohesion of an attribute is the percentage of items occurring in the cluster that contain the attribute, and the overlap is the percentage of items occurring in all other clusters that contain the attribute. An attribute is more strongly correlated with a cluster when it has high cohesion and low overlap. These values provide a means for the grammar engineer to assess the relevance of the attribute to the cluster. An additional feature of this expanded cluster view is that hovering over an attribute from the exemplar item causes all items in the cluster that contain the attribute to be highlighted.

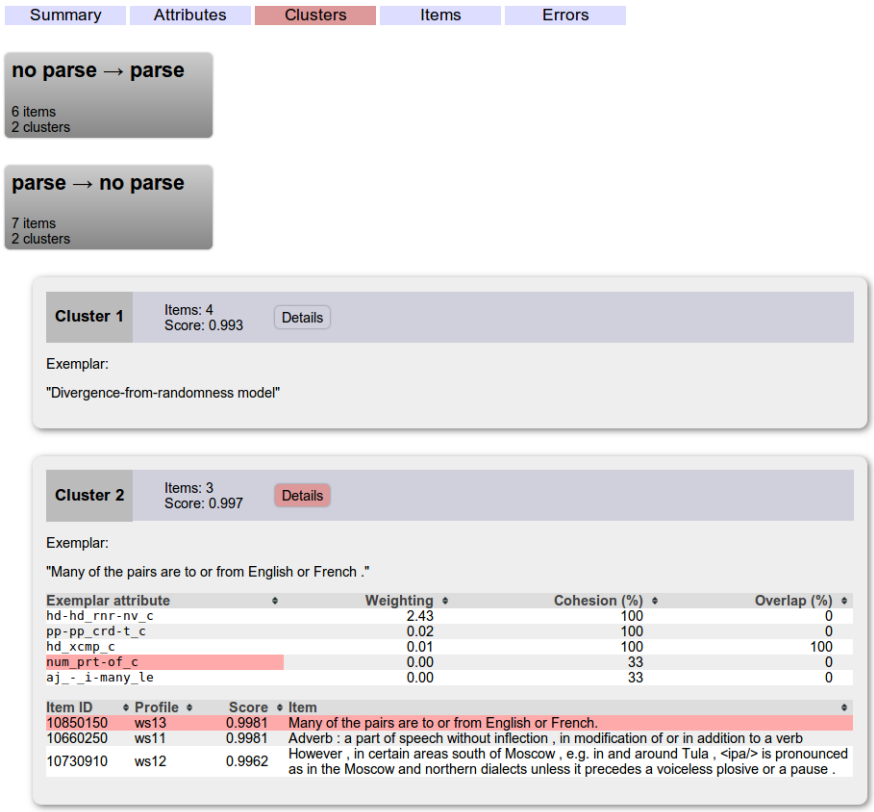

Fig. 6 The Clusters page: shows clusters generated by gDelta across the different categories of parse change. Each cluster is intended to represent a locus of change in the parse results

4.4 Items and Errors Pages

The Items page and Errors page contain additional information not directly related to the attribute weighting and item clustering signals, that augments the feedback provided by the previous three pages. The Items page contains a table of items from all profiles along with information pertaining to each item, such as the input text of the item, the length of the string, the profile the item originated from, an indication of any changes to the parsability of the item, and the number of readings yielded by the two versions of the grammar. As well as this table being sortable, there is also a series of text boxes supporting filtering on fields, as well as on attributes occurring in each item. All of this allows the grammar engineer to drill down quickly and identify individual items with specific properties of interest. For instance, it is trivial to filter on only those items that no longer parse after the change, and that also contain a specific attribute or set of attributes.

The Errors page is similar in structure to the Items page but lists only those items which gDelta encountered errors with. These mostly tend to be parse failures, such as timeouts and system memory limits being reached, but can also arise

due to gDelta failing to extract attributes from the derivation trees of items which were successfully parsed.

5 Evaluation

The evaluation of the techniques used in the creation of gDelta is not straightforward due to the fact that they do not attempt to produce any kind of measurable judgement of correctness. The output is instead diagnostic in nature, requiring interpreting by the grammar engineer — an inherently subjective process. Ultimately the best indicator of the success of gDelta is whether grammar engineers are able to use the tool to more rapidly gain a sense of how the grammar has been impacted by a change. While we have yet to systematically road-test gDelta in the context of active development of a grammar, we have performed some qualitative evaluation of the underlying techniques. This section outlines this evaluation.

This evaluation was an attempt to gauge the utility of gDelta style feedback by simulating contexts from the grammar development process where gDelta is intended to be used. We performed this evaluation using modifications to the ERG which were known to have resulted in problems that were non-trivial to isolate. Participants were charged with the role of grammar engineer and were thus required to have some experience working with the ERG. This restriction unfortunately meant that we only had three participants in the evaluation. This small group of ersatz grammar engineers were asked to hypothesise about the nature of the problem resulting from each change, based upon the attribute ranking and item clustering output from gDelta, along with a short description of what the change was intended to do. Participants were also asked to describe the reasoning process they used to come to these conclusions so as to provide us with insight into how gDelta's output was being used.

We used three separate changes to the ERG made during its development which were ultimately not incorporated into the grammar. Each change involved modifications to the grammar that resulted in problems not apparent at the grammar profiling stage, requiring the analysis of treebanked items before being identified. For each change, the profiles that the two versions of the grammar were run over were taken from the WeScience treebank. The different grammar breakages used were as follows:

> **Change one:** an attempt to reduce the number of analyses being generated for the noun phrase *three thirty*. The existing problem was that one of the analyses produced by the grammar resulted in such phrases being incorrectly analysed as a determiner followed by a noun. A fix for this was attempted by the modification of the type `reg_or_temp_nom_rel` which then resulted in parse failures for valid noun phrases containing numeric determiners such as *two days*.
> **Change two:** an attempt to improve the generality of preposition phrase modification of nominals by changing the type `n_wh_pro_lexent`. This failed to parse prepositional phrases modifying *wh*-pronouns such as in *Who in this town won?*
> **Change three:** an attempt to expand the coverage of extracted complements by modifying the type `extracted_comp_phrase` to implement a

right-node-raising analysis for examples like *We relied on and hired consultants*. The change introduced a regression for items that contained coordinated phrases headed by prepositions, such as *We wrote books for and with children*.

Given that the aim is to identify the problematic effects of these changes, the participants would ideally be able to identify parts of the grammar where the problem manifested itself, rather than just the modified components of the grammar, which the grammar engineer would already be aware of. It should also be noted that these three changes, which all involve minor changes to a single type, represent relatively simple grammar modifications. When changing an existing analysis (or extending the grammar to cover a new one) potentially complex changes could be made to multiple locations in the grammar. Changes of this kind would offer a more compelling context for evaluation, as the potential for unexpected flow on effects that need untangling would be much greater, and would more closely match the intended use case of gDelta. In future work, we hope to perform a more thorough evaluation of gDelta in the context of active grammar development.

The results of the grammar breakage experiments were encouraging. Participants were all able to at least identify the general nature of the problem introduced, if not the specific location in the grammar. All three participants, for instance, were able to identify that the problem in the third change involved numeric determiners, with both the attribute ranking and clustering visualisations clearly indicating that the `num_det_c` rule was likely to be associated with the regression.

The descriptions of the reasoning processes used by participants were also informative. One finding was that the clustering visualisations were considered to be quite useful. In particular, the top-ranking attributes of the exemplar item for each cluster were cited as being a useful diagnostic aid, along with their respective cohesion and overlap values. The attributes were used to determine the nature of the change being represented by the cluster, while the cohesion and overlap of the attributes were used to determine whether this was actually an informative cluster or not. The text of the exemplar item was useful in determining the function of the attributes found in that item.

6 Obtaining and Using gDelta

gDelta was implemented as a Python program that is run from the command line. Running gDelta requires Python 2.7, and installation of the SciPy module is recommended as this will yield faster clustering. Viewing gDelta's output requires a standards-compliant web browser. gDelta is freely available for use and can be found on the DELPH-IN wiki.[6]

Using gDelta involves the user specifying the two different versions of the grammar and their respective output profiles which are to be compared. The user can specify a range of command line flags to run gDelta with different parameters, such as the number of top parse results from which to extract attributes, and the maximum number of clusters to be tried in the clustering stage. Once run, gDelta produces a directory of HTML files containing the different views outlined in Section 4. This directory contains everything required to view the HTML files,

[6] `http://moin.delph-in.net/GDeltaTop`

meaning that gDelta output can be browsed offline and does not require access to the different grammar versions or the parse output. Further instructions for installation are provided with gDelta's source code.

7 Future Work

The most important avenue of work we hope to pursue is substantial in situ road-testing of gDelta in the context of active development of a grammar. This is needed in order to more concretely assess the purported improvements to the grammar engineering toolchain we have argued gDelta offers. It would also provide an opportunity for identifying improvements to the presentation of gDelta's output, as well as additional data points that could be incorporated to further improve the utility of the tool.

One issue in particular that would benefit from further systematic investigation is the relationship between the utility of the signal provided to grammar engineers, the size of the test suites and the range of domains represented by the test suites, and the kind of modification made to the grammar. It is certainly the case that gDelta will be of limited value when used to investigate changes pertaining to linguistic phenomena that are underrepresented in the test suites used as input. Blindly scaling up the range and size of test suites used (if indeed possible) is not necessarily a desirable solution, as increases to parsing time will add latency to the feedback, with the possibility for negating any efficiency improvements. It would be advantageous then to have more of a sense of the scale of input data required for gDelta to provide helpful results.

Further work on the underlying techniques that drive gDelta could involve looking at the impact on other data structures produced by the parser. gDelta currently only makes use of syntactic output generated for successfully parsed items. Incorporating the semantic analysis returned by the parser would likely be beneficial as this is an important signal which must also be monitored for quality assurance, and it can be the case that changes to a grammar only manifest as effects on the semantic analysis. The techniques outlined in this article could also be adapted to use the data structures found in the parse chart, opening up the possibility for partially parsed items — those not receiving a full spanning parse — to also be used.

In the context of DELPH-IN grammars in particular, a specific kind of flow-on effect that can manifest itself in the diagnostic information that gDelta provides is changes in attributes whose corresponding grammar types were not edited by the grammar engineer but inherit from supertypes which were modified. These kinds of interactions are distinct in that they can be readily identified by consulting the type hierarchy of the grammar. Extending gDelta to extract inheritance relationships between edited types and their descendants would provide an additional signal in the visualisations, allowing the grammar engineer to identify the origin of these particular attribute changes. This signal would also be useful in the context of the item clustering, as it would allow for the identification of clusters corresponding to this kind of flow on effect.

8 Conclusion

In this article we presented gDelta, a tool for providing grammar engineers with improved feedback on the effects of changes made to precision grammars. Existing tools that assist with the grammar engineering process tend to provide either immediate but very high-level feedback, or very low-level incremental feedback. gDelta aims to fill this gap between these extremes in the grammar engineering toolchain, providing a more nuanced picture of the impact of the change earlier on in the grammar engineering cycle. It is also the case that existing tools do not explicitly focus on changes made to a grammar, instead requiring the grammar engineer to isolate effects caused by the changes manually. gDelta, however, provides feedback explicitly on changes between two versions of the grammar. We also outlined two techniques used to drive the visualisations gDelta presents to the user: an attribute weighting algorithm used to highlight grammar attributes from the parser output which were significantly impacted by the change, and a test suite item clustering technique intended to identify related groups of change across items whose parsability was affected by the change to the grammar.

The qualitative evaluation outlined in Section 5 supports our belief that gDelta has the potential to improve the efficiency of the grammar engineering process. During the course of the evaluation we also received positive comments from grammar engineers who helped us with this investigation, indicating that they would be interested in using gDelta.

Acknowledgements We would like to thank Emily Bender, Francis Bond and Chikara Hashimoto for their insights into the grammar engineering development process that contributed towards the development of gDelta. We would especially like to thank Dan Flickinger, who additionally provided us with the modified versions of the ERG for use in the evaluation of gDelta. We would also like to thank the three anonymous reviewers, whose comments greatly improved this article.

References

Baldridge J, Chatterjee S, Palmer A, Wing B (2007) DotCCG and VisCCG: Wiki and programming paradigms for improved grammar engineering with OpenCCG. In: Proceedings of the workshop on Grammar Engineering Across Frameworks (GEAF 2007), Manchester, UK

Bender EM, Flickinger D, Oepen S, Zhang Y (2011) Parser evaluation over local and non-local deep dependencies in a large corpus. In: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, Edinburgh, UK, pp 397–408

Butt M, King TH, Niño ME, Segond F (1999) A Grammar Writer's Cookbook. CSLI Publications, Stanford, USA

Butt M, Dyvik H, King TH, Masuichi H, Rohrer C (2002) The Parallel Grammar Project. In: Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics (COLING 2002), Taipei, Taiwan, pp 1–7

Callmeier U (2002) PET – a platform for experimentation with efficient HPSG processing techniques. In: Oepen S, Flickinger D, Tsujii J, Uszkoreit H (eds) Collaborative Language Engineering, CSLI Publications, Stanford, USA

Crabbé B, Duchier D, Gardent C, Le Roux J, Parmentier Y (2013) XMG: eXtensible MetaGrammar. Computational Linguistics 39(3):591–629

Crouch D, Dalrymple M, Kaplan RM, King TH, Maxwell J, Newman P (2014) XLE documentation. `http://www2.parc.com/isl/groups/nltt/xle/doc/xle_toc.html`, Palo Alto Research Center

Dalrymple M (2001) Lexical Functional Grammar. Academic Press, New York, USA
Dost A, King TH (2009) Using large-scale parser output to guide grammar development. In: Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009), Suntec, Singapore, pp 63–70
Flickinger D (2002) On building a more efficient grammar by exploiting types. In: Oepen S, Flickinger D, Tsujii J, Uszkoreit H (eds) Collaborative Language Engineering, Stanford: CSLI Publications, pp 1–17
Gardent C, Narayan S (2012) Error mining on dependency trees. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, Jeju Island, Korea, pp 592–600
Goodman M, Bond F (2009) Using generation for grammar analysis and error detection. In: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, Suntec, Singapore, pp 109–112
Guillaume B, Perrier G (2009) Interaction grammars. Research on Language and Computation 7(2-4):171–208
Joshi AK, Schabes Y (1997) Tree-adjoining grammars. In: Rozenberg G, Salomaa A (eds) Handbook of Formal Languages, vol 3, Springer, Berlin, Germany, pp 69–124
de Kok D, Ma J, van Noord G (2009) A generalized method for iterative error mining in parsing results. In: Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009), Suntec, Singapore, pp 71–79
Müller S (2013) The CoreGram project: A brief overview and motivation. In: Proceedings of the Workshop on High-level Methodologies for Grammar Engineering (HMGE 2013), Düsseldorf, Germany, pp 93–104
van Noord G (2004) Error mining for wide-coverage grammar engineering. In: Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL 2004), Barcelona, Spain, pp 446–453
Oepen S, Carroll J (2000) Parser engineering and performance profiling. Natural Language Engineering 6(1):81–97
Oepen S, Flickinger DP (1998) Towards systematic grammar profiling test suite technology ten years after. Natural Language Engineering, Special Issue on Evaluation 12:411–436
Oepen S, Netter K, Klein J (1997) TSNLP — Test Suites for Natural Language Processing. In: Nerbonne J (ed) Linguistic Databases, CSLI Publications, Stanford, USA, pp 13–36
Oepen S, Toutanova K, Shieber S, Manning C, Flickinger D, Brants T (2002) The LinGO Redwoods treebank. Motivation and preliminary applications. In: Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), Taipei, Taiwan, pp 1–5
Pollard C, Sag IA (1994) Head-driven Phrase Structure Grammar. University of Chicago Press, Chicago, USA
Rousseeuw PJ (1987) Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20:53–65
Sagot B, de La Clergerie E (2006) Error mining in parsing results. In: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia, pp 329–336
Siegel M (2000) HPSG analysis of Japanese. In: Wahlster W (ed) Verbmobil: Foundations of speech-to-speech translation, Springer, Berlin, Germany, pp 264–279
Steedman M (2000) The syntactic process. MIT Press, Cambridge, USA
Tanaka Y (2001) Compilation of a multilingual parallel corpus. In: Proceedings of PACLING 2001, Kitakyushu, Japan, pp 265–268
Waterman SA (2009) Distributed parse mining. In: Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009), Boulder, USA, pp 56–64
Ytrestøl G, Flickinger D, Oepen S (2009) Extracting and annotating Wikipedia sub-domains – towards a new escience community resource. In: Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theory, Groningen, Netherlands, pp 185–197
Zhang Y, Oepen S, Carroll J (2007) Efficiency in unification-based n-best parsing. In: Proceedings of the 10th International Conference on Parsing Technologies, Prague, Czech Republic, pp 48–59