Using Generation for Grammar Analysis and Error Detection

Michael Wayne Goodman*
University of Washington
Dept. of Linguistics
Box 354340 Seattle, WA 98195, USA
goodmami@u.washington.edu

Francis Bond
NICT Language Infrastructure Group
3-5 Hikaridai, Seika-cho, Sōraku-gun,
Kyoto, 619-0289 Japan
bond@ieee.org

Abstract

We demonstrate that the bidirectionality of deep grammars, allowing them to generate as well as parse sentences, can be used to automatically and effectively identify errors in the grammars. The system is tested on two implemented HPSG grammars: Jacy for Japanese, and the ERG for English. Using this system, we were able to increase generation coverage in Jacy by 18% (45% to 63%) with only four weeks of grammar development.

1 Introduction

Linguistically motivated analysis of text provides much useful information for subsequent processing. However, this is generally at the cost of reduced coverage, due both to the difficulty of providing analyses for all phenomena, and the complexity of implementing these analyses. In this paper we present a method of identifying problems in a deep grammar by exploiting the fact that it can be used for both parsing (interpreting text into semantics) and generation (realizing semantics as text). Since both parsing and generation use the same grammar, their performance is closely related: in general improving the performance or cover of one direction will also improve the other. (Flickinger, 2008)

The central idea is that we test the grammar on a full round trip: parsing text to its semantic representation and then generating from it. In general, any sentence where we cannot reproduce the original, or where the generated sentence significantly differs from the original, identifies a flaw in the grammar, and with enough examples we can pinpoint the grammar rules causing these problems. We call our system **Egad**, which stands for Erroneous Generation Analysis and Detection.

*This research was carried out while visiting NICT.

2 Background

This work was inspired by the error mining approach of van Noord (2004), who identified problematic input for a grammar by comparing sentences that parsed and those that didn't from a large corpus. Our approach takes this idea and further applies it to generation. We were also inspired by the work of Dickinson and Lee (2008), whose "variation n-gram method" models the likelihood a particular argument structure (semantic annotation) is accurate given the verb and some context.

We tested **Egad** on two grammars: Jacy (Siegel, 2000), a Japanese grammar and the English Resource Grammar (ERG) (Flickinger, 2000, 2008) from the DELPH-IN[1] group. Both grammars are written in the Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) framework, and use Minimal Recursion Semantics (MRS) (Copestake et al., 2005) for their semantic representations. The Tanaka Corpus (Tanaka, 2001) provides us with English and Japanese sentences.

The specific motivation for this work was to increase the quality and coverage of generated paraphrases using Jacy and the ERG. Bond et al. (2008) showed they could improve the performance of a statistical machine translation system by training on a corpus that included paraphrased variations of the English text. We want to do the same with Japanese text, but Jacy was not able to produce paraphrases as well (the ERG had 83% generation coverage, while Jacy had 45%) Improving generation would also greatly benefit X-to-Japanese machine translation tasks using Jacy.

2.1 Concerning Grammar Performance

There is a difference between the theoretical and practical power of the grammars. Sometimes the

[1]Deep Linguistic Processing with HPSG Initiative – see http://www.delph-in.net for background information, including the list of current participants and pointers to available resources and documentation

Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, pages 109–112,
Suntec, Singapore, 4 August 2009.

parser or generator can reach the memory (i.e. edge) limit, resulting in a valid result not being returned. Also, we only look at the top-ranked[2] parse and the first five generations for each item. This is usually not a problem, but it could cause **Egad** to report false positives.

HPSG grammars are theoretically symmetric between parsing and generation, but in practice this is not always true. For example, to improve performance, semantically empty lexemes are not inserted into a generation unless a "trigger-rule" defines a context for them. These trigger-rules may not cover all cases.

3 Grammar Analysis

When analyzing a grammar, **Egad** looks at all input sentences, parses, and generations processed by the grammar and uses the information therein to determine characteristics of these items. These characteristics are encoded in a vector that can be used for labeling and searching items. Some characteristics are useful for error mining, while others are used for grammar analysis.

3.1 Characteristic Types

Egad determines both general characteristics of an item (parsability and generability), and characteristics comparing parses with generations.

General characteristics show whether each item could: be parsed ("parsable"), generate from parsed semantics ("generable"), generate the original parsed sentence ("reproducible"), and generate other sentences ("paraphrasable").

For comparative characteristics, **Egad** compares every generated sentence to the parsed sentence whence its semantics originated, and determines if the generated sentence uses the same set of lexemes, derivation tree,[3] set of rules, surface form, and MRS as the original.

3.2 Characteristic Patterns

Having determined all applicable characteristics for an item or a generated sentence, we encode the values of those characteristics into a vector. We call this vector a **characteristic pattern**, or CP. An example CP showing general characteristics is:

```
0010 -----
```

The first four digits are read as: the item is parsable, generable, not reproducible, and is paraphrasable. The five following dashes are for comparative characteristics and are inapplicable except for generations.

3.3 Utility of Characteristics

Not all characteristics are useful for all tasks. We were interested in improving Jacy's ability to generate sentences, so we primarily looked at items that were parsable but ungenerable. In comparing generated sentences with the original parsed sentence, those with differing semantics often point to errors, as do those with a different surface form but the same derivation tree and lexemes (which usually means an inflectional rule was misapplied).

4 Problematic Rule Detection

Our method for detecting problematic rules is to train a maximum entropy-based classifier[4] with n-gram paths of rules from a derivation tree as features and characteristic patterns as labels. Once trained, we do feature-selection to look at what paths of rules are most predictive of certain labels.

4.1 Rule Paths

We extract n-grams over **rule paths**, or RPs, which are downward paths along the derivation tree. (Toutanova et al., 2005) By creating separate RPs for each branch in the derivation tree, we retain some information about the order of rule application without overfitting to specific tree structures. For example, Figure 1 is the derivation tree for (1). A couple of RPs extracted from the derivation tree are shown in Figure 2.

(1) 写真 写り が いい
shashin-utsuri-ga ii
picture-taking-NOM good
(X is) good at taking pictures.

4.2 Building a Model

We build a classification model by using a parsed or generated sentence's RPs as features and that sentence's CP as a label. The set of RPs includes n-grams over all specified values of N. The labels are, to be more accurate, regular expressions of

[2] Jacy and the ERG both have parse-ranking models.

[3] In comparing the derivation trees, we only look at phrasal nodes. Lexemes and surface forms are not compared.

[4] We would like to look at using different classifiers here, such as Decision Trees. We initially chose MaxEnt because it was easy to implement, and have since had little motivation to change it because it produced useful results.

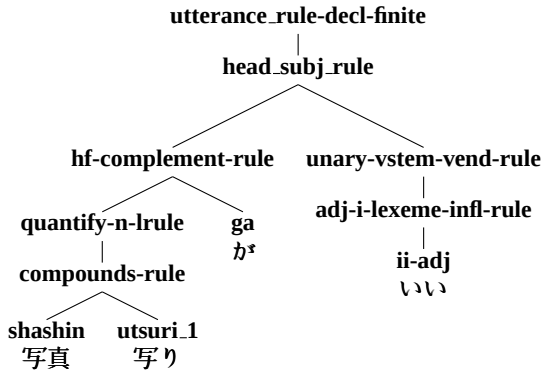

Figure 1: Derivation tree for (1)

quantify-n-lrule → *compounds-rule* → *shashin*
quantify-n-lrule → *compounds-rule* → *utsuri_1*

Figure 2: Example RPs extracted from Figure 1

CPs and may be fully specified to a unique CP or generalize over several.[5] The user can weight the RPs by their N value (e.g. to target unigrams).

4.3 Finding Problematic Rules

After training the model, we have a classifier that predicts CPs given a set of RPs. What we want, however, is the RP most strongly associated with a given CP. The classifier we use provides an easy method to get the score a given feature has for some label. We iterate over all RPs, get their score, then sort them based on the score. To help eliminate redundant results, we exclude any RP that either subsumes or is subsumed by a previous (i.e. higher ranked) RP.

Given a CP, the RP with the highest score should indeed be the one most closely associated to that CP, but it might not lead to the greatest number of items affected. Fixing the second highest ranked RP, for example, may improve more items than fixing the top ranked one. To help the grammar developer decide the priority of problems to fix, we also output the count of items observed with the given CP and RP.

5 Results and Evaluation

We can look at two sets of results: how well **Egad** was able to analyze a grammar and detect errors, and how well a grammar developer could use **Egad** to fix a problematic grammar. While the latter is also influenced by the skill of the grammar developer, we are interested in how well **Egad** points to the most significant errors, and how it can help reduce development time.

5.1 Error Mining

Table 1 lists the ten highest ranked RPs associated with items that could parse but could not generate in Jacy. Some RPs appear several times in different contexts. We made an effort to decrease the redundancy, but clearly this could be improved.

From this list of ten problematic RPs, there are four unique problems: quantify-n-lrule (noun quantification), no-nspec (noun specification), to-comp-quotarg (と *to* quotative particle), and te-adjunct (verb conjugation). The extra rules listed in each RP show the context in which each problem occurs, and this can be informative as well. For instance, *quantify-n-lrule* occurs in two primary contexts (above *compounds-rule* and *nominal-numcl-rule*). The symptoms of the problem occur in the interation of rules in each context, but the source of the problem is *quantify-n-lrule*.

Further, the problems identified are not always lexically marked. *quantify-n-lrule* occurs for all bare noun phrases (ie. without determiners). This kind of error cannot be accurately identified by using just word or POS n-grams, we need to use the actual parse tree.

5.2 Error Correction

Egad greatly facilitated our efforts to find and fix a wide variety of errors in Jacy. For example, we restructured semantic predicate hierarchies, fixed noun quantification, allowed some semantically empty lexemes to generate in certain contexts, added pragmatic information to distinguish between politeness levels in pronouns, allowed imperatives to generate, allowed more constructions for numeral classifiers, and more.

Egad also identified some issues with the ERG: both over-generation (an under-constrained inflectional rule) and under-generation (sentences with the construction *take* $\{care|charge|\ldots\}$ *of* were not generating).

5.3 Updated Grammar Statistics

After fixing the most significant problems in Jacy (outlined in Section 5.2) as reported by **Egad**, we obtained new statistics about the grammar's coverage and characteristics. Table 2 shows the original and updated general statistics for Jacy. We increased generability by 18%, doubled reproducibility, and increased paraphrasability by 17%.

[5] For example, `/0010 -----/` is fully specified. `/00.. -----/` marginalizes two general characteristics

Score	Count	Rule Path N-grams
1.42340952569648	109	hf-complement-rule → quantify-n-lrule → compounds-rule
0.960090299833317	54	hf-complement-rule → quantify-n-lrule → nominal-numcl-rule → head-specifier-rule
0.756227560530811	63	head-specifier-rule → hf-complement-rule → no-nspec → "の"
0.739668926140179	62	hf-complement-rule → head-specifier-rule → hf-complement-rule → no-nspec
0.739090261637851	22	hf-complement-rule → hf-adj-i-rule → quantify-n-lrule → compounds-rule
0.694215264789286	36	hf-complement-rule → hf-complement-rule → to-comp-quotarg → "と"
0.676244980660372	82	vstem-vend-rule → te-adjunct → "て"
0.617621482523537	26	hf-complement-rule → hf-complement-rule → to-comp-varg → "と"
0.592260546433334	36	hf-adj-i-rule → hf-complement-rule → quantify-n-lrule → nominal-numcl-rule
0.564790702894285	62	quantify-n-lrule → compounds-rule → vn2n-det-lrule

Table 1: Top 10 RPs for ungenerable items

	Original	Modified
Parsable	82%	83%
Generable	45%	63%
Reproducible	11%	22%
Paraphrasable	44%	61%

Table 2: Jacy's improved general statistics

As an added bonus, our work focused on improving generation also improved parsability by 1%. Work is now continuing on fixing the remainder of the identified errors.

6 Future Work

In future iterations of **Egad**, we would like to expand our feature set (e.g. information from failed parses), and make the system more robust, such as replacing lexical-ids (specific to a lexeme) with lexical-types, since all lexemes of the same type should behave identically. A more long-term goal would allow **Egad** to analyze the internals of the grammar and point out specific features within the grammar rules that are causing problems. Some of the errors detected by **Egad** have simple fixes, and we believe there is room to explore methods of automatic error correction.

7 Conclusion

We have introduced a system that identifies errors in implemented HPSG grammars, and further finds and ranks the possible sources of those problems. This tool can greatly reduce the amount of time a grammar developer would spend finding bugs, and helps them make informed decisions about which bugs are best to fix. In effect, we are substituting cheap CPU time for expensive grammar developer time. Using our system, we were able to improve Jacy's absolute generation coverage by 18% (45% to 63%) with only four weeks of grammar development.

8 Acknowledgments

Thanks to NICT for their support, Takayuki Kuribayashi for providing native judgments, and Marcus Dickinson for comments on an early draft.

References

Francis Bond, Eric Nichols, Darren Scott Appling, and Michael Paul. 2008. Improving statistical machine translation by paraphrasing the training data. In *International Workshop on Spoken Language Translation*, pages 150–157. Honolulu.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal Recursion Semantics. An introduction. *Research on Language and Computation*, 3(4):281–332.

Markus Dickinson and Chong Min Lee. 2008. Detecting errors in semantic annotation. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*. Marrakech, Morocco.

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28. (Special Issue on Efficient Processing with HPSG).

Dan Flickinger. 2008. The English resource grammar. Technical Report 2007-7, LOGON, `http://www.emmtee.net/reports/7.pdf`. (Draft of 2008-11-30).

Carl Pollard and Ivan A. Sag. 1994. *Head Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Melanie Siegel. 2000. HPSG analysis of Japanese. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, pages 265 – 280. Springer, Berlin, Germany.

Yasuhito Tanaka. 2001. Compilation of a multilingual parallel corpus. In *Proceedings of PACLING 2001*, pages 265–268. Kyushu. (`http://www.colips.org/afnlp/archives/pacling2001/pdf/tanaka.pdf`).

Kristina Toutanova, Christopher D. Manning, Dan Flickinger, and Stephan Oepen. 2005. Stochastic HPSG parse disambiguation using the redwoods corpus. *Research on Language and Computation*, 3(1):83–105.

Gertjan van Noord. 2004. Error mining for wide-coverage grammar engineering. In *42nd Annual Meeting of the Association for Computational Linguistics: ACL-2004*. Barcelona.