Designing Testsuites for Grammar-based Systems in Applications

Valeria de Paiva
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304 USA
valeria.paiva@gmail.com

Tracy Holloway King
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304 USA
thking@parc.com

Abstract

In complex grammar-based systems, even small changes may have an unforeseeable impact on overall system performance. Regression testing of the system and its components becomes crucial for the grammar engineers developing the system. As part of this regression testing, the testsuites themselves must be designed to accurately assess coverage and progress and to help rapidly identify problems. We describe a system of passage-query pairs divided into three types of phenomenon-based testsuites (sanity, query, basic correct). These allow for rapid development and for specific coverage assessment. In addition, real-world testsuites allow for overall performance and coverage assessment. These testsuites are used in conjunction with the more traditional representation-based regression testsuites used by grammar engineers.

1 Introduction

In complex grammar-based systems, even small changes may have an unforeseeable impact on overall system performance.[1] Systematic regression testing helps grammar engineers to track progress, and to recognize and correct shortcomings in linguistic rule sets. It is also an essential tool for assessing overall system status in terms of task and runtime performance.

As discussed in (Chatzichrisafis et al., 2007), regression testing for grammar-based systems involves two phases. The first includes systematic testing of the grammar rule sets during their development. This is the part of regression testing that grammar engineers are generally most familiar with. The second phase involves the deployment of the grammar in a system and the regression testing of the grammar as a part of the whole system. This allows the grammar engineer to see whether changes have any effect on the system, positive or negative. In addition, the results of regression testing in the system allow a level of abstraction away from the details of the grammar output, which can ease maintenance of the regression testsuites so that the grammar engineers do not need to change the gold standard annotation every time an intermediate level of representation changes.

In this paper, we focus on the design of testsuites for grammar-based systems, using a question-answering system as a model. In particular, we are interested in what types of testsuites allow for rapid development and efficient debugging.

1.1 The Question-Answering System

To anchor the discussion, we focus on regression testsuites designed for a grammar-based question-answering system (Bobrow et al., 2007). The Bridge system uses the XLE (Crouch et al., 2008) parser to produce syntactic structures and then the XLE ordered rewrite system to produce linguistic semantics (Crouch and King, 2006) and abstract knowledge representations. Abstract knowledge representations for passages and queries are processed by an entailment and contradiction detection system which determines whether the query is

[1] We would like to thank Rowan Nairn for his design and implementation of the regression platform that runs these testsuites. We would also like to thank the PARC Natural Language Theory and Technology group for their work with these testsuites and their comments on this paper.

entailed, contradicted, or neither by the passage.

Entailment and contradiction detection between passages and queries is a task well suited to regression testing. There are generally only two or three possible answers given a passage and a query: entails, contradicts or neither (or in the looser case: relevant or irrelevant). *Wh*-questions (section 5.1) receive a YES answer if an alignment is found between the *wh*-word in the query and an appropriate part of the passage representation; in this case, the proposed alignment is returned as well as the YES answer. This is particularly important for *who* and *what* questions where more than one entity in the passage might align with the *wh*-word.

From the standpoint of regression testing, two important aspects of the question-answering application are:

(1) The correct answer for a given pair is independent of the representations used by the system and even of which system is used.

(2) The passage-query pairs with answers can be constructed by someone who does not know the details of the system.

The first aspect means that even drastic changes in representation will not result in having to update the regression suites. This contrasts sharply with regressions run against representative output which require either that the gold standard be updated or that the mapping from the output to that standard be updated. The second aspect means that externally developed testsuites (e.g. FraCaS (Cooper et al., 1996), Pascal RTE (Sekine et al., 2007)) can easily be incorporated into the regression testing and that grammar engineers can rapidly add new testsuites, even if they do not have experience with the internal structure of the system. These aspects also mean that such passage-query pairs can be used for cross-system comparisons of coverage (Bos, 2008).

1.2 Testsuite Types

In the regression testsuites designed for the question-answering system, the passage-query pair testsuites are divided into two main types: those that focus on single phenomena (section 2) and those that use real-world passages (section 3). The phenomenon-based testsuites allow the grammar engineer to track the behavior of the system with respect to a given construction, such as implicativity, noun-noun compounds, temporal expressions, or comparatives. In contrast, the real-world passages allow the grammar engineer to see how the system will behave when applied to real data, including data which the system will encounter in applications. Such sentences tend to stress the system in terms of basic performance (e.g. efficiency and memory requirements for processing of long sentences) and in terms of interactions of different phenomena (e.g. coordination ambiguity interacting with implicativity).

In addition to the passage-query pairs, the system includes regression over representations at several levels of analysis (section 4). These are limited in number, focus only on core phenomena, and are not gold standard representations but instead the best structure of the ones produced. These are used to detect whether unintentional changes were introduced to the representations (e.g. new features were accidentally created).

2 Phenomenon Sets

Real-world sentences involve analysis of multiple interacting phenomena. Longer sentences tend to have more diverse sets of phenomena and hence a higher chance of containing a construction that the system does not handle well. This can lead to frustration for grammar engineers trying to track progress; fixing a major piece of the system can have little or no effect on a testsuite of real-world examples. To alleviate this frustration, we have extensive sets of hand-crafted test examples that are focused as much as possible on single phenomenon. These include externally developed testsuites such as the FraCaS (Cooper et al., 1996) and HP testsuites (Nerbonne et al., 1988). Focused testsuites are also good for quickly diagnosing problems. If all the broken examples are in the deverbal testsuite, for example, it gives grammar engineers a good idea of where to look for bugs.

The majority of the testsuites are organized by syntactic and semantic phenomena and are designed to test all known variants of that phenomenon (see (Cohen et al., 2008) on the need to use testsuites designed to test system coverage as well as real-world corpora). For the question-answering system, these include topics such as anaphora, appositives, copulars, negation, deverbal nouns and adjectives, implicatives and factives, temporals, cardinality and quantifiers, comparatives, possessives, context introducing nouns, and pertainyms. These categories align with many of

those cited by (Bos, 2008) in his discussion of semantic parser coverage. Some example passage-query pairs for deverbal nouns are shown in (3).

(3) a. **P:** Ed's abdication of the throne was welcome.
Q: Ed abdicated the throne.
A: YES

b. **P:** Ed's abdication was welcome.
Q: Ed abdicated.
A: YES

c. **P:** Ed is an abdicator.
Q: Ed abdicated.
A: YES

Each of the phenomena has three sets of testsuites associated with it. Sanity sets (section 2.1) match a passage against itself. The motivation behind this is that a passage should generally entail itself and that if the system cannot capture this entailment, something is wrong. Query sets (section 2.2) match the passage against query versions of the passage. The simplest form of this is to have a polarity question formed from the passage. More complex versions involve negative polarity questions, questions with different adjuncts or argument structures, and questions with synonyms or antonyms. Basic correct sets (section 2.3) are selected passage-query pairs in which the system is known to obtain the correct answer for the correct reason. The idea behind these sets is that they can be run immediately by the grammar engineer after making any changes and the results should be 100% correct: any mistakes indicates a problem introduced by the grammar engineer's changes.

2.1 Sanity Sets

The entailment and contradiction detection part of the system is tested in isolation by matching queries against themselves. Some example sanity pairs from the copula testsuite are shown in (4).

(4) a. **P:** A boy is tall.
Q: A boy is tall.
A: YES

b. **P:** A girl was the hero.
Q: A girl was the hero.
A: YES

c. **P:** The boy is in the garden.
Q: The boy is in the garden.
A: YES

d. **P:** The boy is not in the garden.
Q: The boy is not in the garden.
A: YES

Note that queries in the question-answering system do not have to be syntactically interrogative. This allows the sanity pairs to be processed by the same mechanism that processes passage-query pairs with syntactically interrogative queries.

The sanity check testsuites are largely composed of simple, hand-crafted examples of all the syntactic and semantic patterns that the system is known to cover. This minimal check ensures that at least identical representations trigger an entailment.

2.2 Query Sets

The query sets form the bulk of the regression sets. The query sets comprise passages of the types found in the sanity sets, but with more complex queries. The simplest form of these is to form the polarity question from the passage, as in (5). More complex queries can be formed by switching the polarity from the passage to the query, as in (6).

(5) a. **P:** A boy is tall.
Q: Is a boy tall?
A: YES

b. **P:** A girl was the hero.
Q: Was a girl the hero?
A: YES

(6) **P:** The boy is not in the garden.
Q: Is the boy in the garden?
A: NO

To form more complex pairs, adjuncts and argument structure can be altered from the passage to the query. These have to be checked carefully to ensure that the correct answer is coded for the pair since entailment relations are highly sensitive to such changes. Some examples are shown in (7). Alternations such as those in (7c) are crucial for testing implicativity, which plays a key role in question answering.

(7) a. **P:** An older man hopped.
Q: A man hopped.
A: YES

b. **P:** John broke the box.
Q: The box broke.
A: YES

c. **P:** Ed admitted that Mary arrived.
Q: Mary arrived.
A: YES

A similar type of alteration of the query is to substitute synonyms for items in the passage, as in (8). This is currently done less systematically in the testsuites but helps determine lexical coverage.

(8) a. **P:** Some governments ignore historical facts.
Q: Some governments ignore the facts of history.
A: YES

b. **P:** The boys bought some candy.
Q: The boys purchased some candy.
A: YES

In addition to the testsuites created by the question-answering system developers, the query sets include externally developed pairs, such as those created for FraCaS (Cooper et al., 1996). These testsuites also involve handcrafted passage-query pairs, but the fact that they were developed outside of the system helps to detect gaps in system coverage. In addition, some of the FraCaS pairs involve multi-sentence passages. Since the sentences in these passages are very short, they are appropriate for inclusion in the phenomenon-based testsuites. Some externally developed testsuites such as the HP testsuite (Nerbonne et al., 1988) do not involve passage-query pairs but the same techniques used by the grammar engineers to create the sanity and the query sets are applied to these testsuites as well.

2.3 Basic Correct Sets

A subset of the query sets described above are used to form a core set of basic correct testsuites. These testsuites contain passage-query pairs that the developers have determined the system is answering correctly for the correct reason.

Since these testsuites are run each time the grammar engineer makes a change to the system before checking the changes into the version control repository, it is essential that the basic correct testsuites can be run quickly. Each pair is processed rapidly because the query sets are composed of simple passages that focus on a given phenomenon. In addition, only one or two representatives of any given construction is included in the basic correct set; that is, the sanity sets and query sets may contain many pairs testing copular constructions with adjectival complements, but only a small subset of these are included in the basic correct set. In the question-answering system, $\sim$375 passage-query pairs are in the basic correct sets; it takes less than six minutes to run the full set on standard machines. In addition, since the basic correct sets are divided by phenomena, developers can first run those testsuites which relate directly to the phenomena they have been working on.

Examining the basic correct sets gives an overview of the expected base coverage of the system. In addition, since all of the pairs are working for the correct reason when they are added to the basic correct set, any breakage is a sign that an error has been introduced into the system. It is important to fix these immediately so that grammar engineers working on other parts of the system can use the basic correct sets to assess the impact of their changes on the system.

3 Real-world Sets

The ultimate goal of the system is to work on real-world texts used in the application. So, tests of those texts are important for assessing progress on naturally occurring data. These testsuites are created by extracting sentences from the corpora expected to be used in the run-time system, e.g. newspaper text or the Wikipedia.[2] Queries are then created by hand for these sentences. Once the system is being used by non-developers, queries posed by those users can be incorporated into the testsuites to ensure that the real-world sets have an appropriate range of queries. Currently, the system uses a combination of hand-crafted queries and queries from the RTE data which were hand-crafted, but not by the question-answering system developers. Some examples are shown in (9).

(9) a. **P:** The interest of the automotive industry increases and the first amplifier project, a four-channel output module for the German car manufacturer, Porsche, is finished.
Q: Porsche is a German car manufacturer.
A: YES

b. **P:** The Royal Navy servicemen being held captive by Iran are expected to be freed to-

[2] If the application involves corpora containing ungrammatical input (e.g. email messages), it is important to include both real-world and phenomenon sets for such data.

day.
Q: British servicemen detained
A: YES

c. **P:** "I guess you have to expect this in a growing community," said Mardelle Kean, who lives across the street from John Joseph Famalaro, charged in the death of Denise A. Huber, who was 23 when she disappeared in 1991.
Q: John J. Famalaro is accused of having killed Denise A. Huber.
A: YES

These real-world passages are not generally useful for debugging during the development cycle. However, they serve to track progress over time, to see where remaining gaps may be, and to provide an indication of system performance in applications. For example, the passage-query pairs can be roughly divided as to those using just linguistic meaning, those using logical reasoning, those requiring plausible reasoning, and finally those requiring world knowledge. Although the boundaries between these are not always clear (Sekine et al., 2007), having a rough division helps in guiding development.

4 Regression on Representations

There has been significant work on regression testing of a system's output representations (Nerbonne et al., 1988; Cooper et al., 1996; Lehmann et al., 1996; Oepen et al., 1998; Oepen et al., 2002): designing of the testsuites, running and maintaining them, and tracking the results over time. As mentioned in the previous discussion, for a complex system such as a question-answering system, having regression testing that depends on the performance of the system rather than on details of the representations has significant advantages for development because the regression testsuites do not have to be redone whenever there is a change to the system and because the gold standard items (i.e., the passage-query pairs with answers) can be created by those less familiar with the details of the system.

However, having a small but representative set of banked representations at each major level of system output has proven useful for detecting unintended changes that may not immediately disturb the passage-query pairs.[3] This is especially the case with the sanity sets and the most basic query sets: with these the query is identical to or very closely resembles the passage so that changes to the representation on the passage side will also be in the representation on the query side and hence may not be detected as erroneous by the entailment and contradiction detection.

For the question-answering system, ∼1200 sentences covering basic syntactic and semantic types form a testsuite for representations. The best representation currently produced by the system is stored for the syntax, the linguistic semantics, and the abstract knowledge representation levels. To allow for greater stability over time and less sensitivity to minor feature changes in the rule sets, it is possible to bank only the most important features in the representations may, e.g. the core predicate-argument structure. The banked representations are then compared with the output of the system after any changes are made. Any differences are examined to see whether they are intentional. If they were intended, then new representations need to be banked for the ones that have changed (see (Rosén et al., 2005) for ways to speed up this process by use of discrimants). If the differences were not intended, then the developer knows which constructions were affected by their changes and can more easily determine where in the system the error might have been introduced.

5 Discussion and Conclusions

The testsuites discussed above are continually under development. We believe that the basic ideas behind these testsuites should be applicable to other grammar-based systems used in applications. The passage-query pairs are most applicable to question-answering and search/retrieval systems, but aspects of the approach can apply to other systems.

Some issues that remain for the testsuites discussed above are extending the use of *wh*-questions in passage-query pairs, the division between development and test sets, and the incorporation of context into the testing.

5.1 *Wh*-questions

The testsuites as described have not yet been systematically extended to *wh*-questions. The query

[3] In addition to running regression tests against representations, the syntax, semantics, and abstract knowledge representation have type declarations (Crouch and King, 2008) which help to detect malformed representations.

sets can be easily extended to involve some substitution of *wh*-phrases for arguments and adjuncts in the passage, as in (10).

(10) a. **P:** John broke the box.
Q: Who broke the box?

b. **P:** John broke the box.
Q: What did John break?

c. **P:** John broke the box.
Q: What broke?

d. **P:** John broke the box.
Q: What did John do?

e. **P:** We went to John's party last night.
Q: Who went to John's party?

There is a long-standing issue as to how to evaluate responses to *wh*-questions (see (Voorhees and Tice, 2000a; Voorhees and Tice, 2000b) and the TREC question-answering task web pages for discussion and data). For example, in (10a) most people would agree that the answer should be *John*, although there may be less agreement as to whether *John broke the box*. is an appropriate answer. In (10b) and (10c) there is an issue as to whether the answer should be *box* or *the box* and how to assess partial answers. This becomes more of an issue as the passages become more complicated, e.g. with heavily modified nominals that serve as potential answers. While for (10d) the passage is a good answer to the question, for (10e) presumably the answer should be a list of names, not simply "we". Obtaining such lists and deciding how complete and appropriate they are is challenging. Since most question-answering systems are not constrained to polarity questions, it is important to assess performance on *wh*-questions as the system develops. Other, even more complicated questions, for example *how to* questions are also currently out of the scope of our testsuites.

5.2 Development vs. Testing

For development and evaluation of systems, testsuites are usually divided into development sets, which the system developers examine in detail, and test sets, which represent data unseen by the developers.[4] To a limited extent, the real-world sets serve as a form of test set since they reflect the performance of the system on real data and are often not examined in detail for why any given pair fails to parse. However, the testsuites described above are all treated as development sets. There are no reserved phenomenon-based testsuites for blind testing of the system's performance on each phenomenon, although there are real-world testsuites reserved as test sets.

If a given testsuite was created all at once, a random sampling of it could be held out as a test set. However, since there are often only a few pairs per construction or lexical item, it is unclear whether this approach would give a fair view of system coverage. In addition, for rule-based systems such as the syntax and semantics used in the question-answering system, the pairs are often constructed based on the rules and lexicons as they were being developed. As such, they more closely match the coverage of the system than if it were possible to randomly select such pairs from external sources.

As a system is used in an application, a test set of unseen, application-specific data becomes increasingly necessary. Such sets can be created from the use of the application: for example, queries and returned answers with judgments as to correctness can provide seeds for test sets, as well as for extending the phenomenon-based and real-world development testsuites.

5.3 Context

The real sentences that a question-answering system would use to answer questions appear in a larger textual and metadata context. This context provides information as to the resolution of pronouns, temporal expressions such as *today* and *this morning*, ellipsis, etc. The passage-query pairs in the testsuites do not accurately reflect how well the system handles the integration of context. Small two sentence passages can be used to, for example, test anaphora resolution, as shown in (11).

(11) **P:** Mary hopped. Then, she skipped.
Q: Did Mary skip?
A: YES

Even in this isolated example, the answer can be construed as being UNKNOWN since it is possible, although unlikely, that *she* resolves to some other entity. This type of problem is pervasive in using

[4] The usual division is between training, development, and test sets, with the training set generally being much larger than the development and test sets. For rule based systems, the training/development distinction is often irrelevant, and so a distinction is made between those sets used in the development of the system and those unseen sets used to test and evaluate the system's performance.

simple passage-query pairs for system regression testing.

A further issue with testing phenomena linked to context, such as anaphora resolution, is that they are usually very complex and can result in significant ambiguity. When used on real-world texts, efficiency can be a serious issue which this type of more isolated testing does not systematically explore. As a result of this, the anaphora testsuites must be more carefully constructed to take advantage of isolated, simpler pairs when possible but to also contain progressively more complicated examples that eventually become real-world pairs.

5.4 Summary Conclusions

In complex grammar-based systems, even small changes may have an unforeseeable impact on system performance. Regression testing of the system and its components becomes crucial for the grammar engineers developing the system.

A key part of regression testing is the testsuites themselves, which must be designed to accurately assess coverage and progress and to help to rapidly identify problems. For broad-coverage grammars, such as those used in open domain applications like consumer search and question answering, testsuite design is particularly important to ensure adequate coverage of basic linguistic (e.g. syntactic and semantic) phenomena as well as application specific phenomena (e.g. interpretation of markup, incorporation of metadata).

We described a system of passage-query pairs divided into three types of phenomenon-based testsuites (sanity, query, basic correct). These allow for rapid development and specific coverage assessment. In addition, real-world testsuites allow for overall performance and coverage assessment. More work is needed to find a systematic way to provide "stepping stones" in terms of complexity between phenomenon-based and real-world testsuites.

These testsuites are used in conjunction with the more traditional representation-based regression testsuites used by grammar engineers. These representation-based testsuites use the same phenomenon-based approach in order to assess coverage and pinpoint problems as efficiently as possible.

References

Bobrow, Daniel G., Bob Cheslow, Cleo Condoravdi, Lauri Karttunen, Tracy Holloway King, Rowan Nairn, Valeria de Paiva, Charlotte Price, and Annie Zaenen. 2007. PARC's bridge and question answering system. In King, Tracy Holloway and Emily M. Bender, editors, *Grammar Engineering Across Frameworks*, pages 46–66. CSLI Publications.

Bos, Johan. 2008. Let's not argue about semantics. In *Proceedings of LREC*.

Chatzichrisafis, Nikos, Dick Crouch, Tracy Holloway King, Rowan Nairn, Manny Rayner, and Marianne Santaholma. 2007. Regression testing for grammar-based systems. In King, Tracy Holloway and Emily M. Bender, editors, *Proceedings of the Grammar Engineering Across Frameworks (GEAF07) Workshop*, pages 128–143. CSLI Publications.

Cohen, K. Bretonnel, William A. Baumgartner Jr., and Lawrence Hunter. 2008. Software testing and the naturally occurring data assumption in natural language processing. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 23–30. Association for Computational Linguistics.

Cooper, Robin, Dick Crouch, Jan van Eijck, Chris Fox, Josef van Genabith, Jan Jaspars, Hans Kamp, David Milward, Manfred Pinkal, Massimo Poesio, and Steve Pulman. 1996. Using the framework. FraCas: A Framework for Computational Semantics (LRE 62-051).

Crouch, Dick and Tracy Holloway King. 2006. Semantics via f-structure rewriting. In Butt, Miriam and Tracy Holloway King, editors, *LFG06 Proceedings*, pages 145–165. CSLI Publications.

Crouch, Dick and Tracy Holloway King. 2008. Typechecking in formally non-typed systems. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 3–4. Association for Computational Linguistics.

Crouch, Dick, Mary Dalrymple, Ron Kaplan, Tracy King, John Maxwell, and Paula Newman. 2008. XLE documentation. http://www2.parc.com/isl/groups/nltt/xle/doc/.

Lehmann, Sabine, Stephan Oepen, Sylvie Regnier-Prost, Klaus Netter, Veronika Lux, Judith Klein, Kirsten Falkedal, Frederik Fouvry, Dominique Estival, Eva Dauphin, Hervé Compagnion, Judith Baur, Lorna Balkan, and Doug Arnold. 1996. TSNLP — Test Suites for Natural Language Processing. In *Proceedings of COLING 1996*.

Nerbonne, John, Dan Flickinger, and Tom Wasow. 1988. The HP Labs natural language evaluation tool. In *Proceedings of the Workshop on Evaluation of Natural Language Processing Systems*.

Oepen, Stephan, Klaus Netter, and Judith Klein. 1998. TSNLP — Test Suites for Natural Language Processing. In Nerbonne, John, editor, *Linguistic Databases*, pages 13–36. CSLI.

Oepen, Stephan, Dan Flickinger, Kristina Toutanova, and Chris D. Manning. 2002. LinGO Redwoods. a rich and dynamic treebank for HPSG. In *Proceedings of The First Workshop on Treebanks and Linguistic Theories*, pages 139–149.

Rosén, Victoria, Koenraad de Smedt, Helge Dyvik, and Paul Meurer. 2005. TREPIL: Developing methods and tools for multilevel treebank construction. In *Proceedings of The Fourth Workshop on Treebanks and Linguistic Theories*.

Sekine, Satoshi, Kentaro Inui, Ido Dagan, Bill Dolan, Danilo Giampiccolo, and Bernardo Magnini, editors. 2007. *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*. Association for Computational Linguistics, Prague, June.

Voorhees, Ellen and Dawn Tice. 2000a. Building a question answering test collection. In *Proceedings of SIGIR-2000*, pages 200–207.

Voorhees, Ellen and Dawn Tice. 2000b. The TREC-8 question answering track evaluation. In *Proceedings 8th Text REtrieval Conference (TREC-8)*, pages 83–105.