Regression Testing For Grammar-Based Systems

Nikos Chatzichrisafis[1], Dick Crouch[3], Tracy Holloway King[2], Rowan Nairn[2], Manny Rayner[1,3], Marianne Santaholma[1]

(1) University of Geneva, TIM/ISSCO
(2) Palo Alto Research Center
(3) Powerset, Inc.

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

In complex grammars, even small changes may have an unforeseeable impact on overall system performance. As grammar based systems are increasingly deployed for industrial and other large-scale applications, it is imperative to have systematic regression testing in place. Systematic testing in grammar-based systems serves several purposes. It helps developers to track progress, and to recognize and correct shortcomings in linguistics rules sets. It is also an essential tool for assessing overall system status in terms of task and runtime performance.

This paper describes best practices in two closely related regression testing frameworks used in grammar-based systems: MedSLT, a spoken language translation system based on the Regulus platform, and a search and question answering system based on PARC's XLE syntax-semantics parser.

1 Introduction

Regression testing is an important part of all software development, including large-scale, application-oriented grammars. Similar to regression testing in other systems, regression testing in the context of grammar development ascertains the correctness of the code and its output alongside important systems issues such as how quickly the grammar and system run. This helps to ensure systematic development both of the grammar and the system using the grammar.

In the context of grammar engineering, this applies in particular to linguistic rule-sets and the compilers and interpreters used to process them. Systematic testing identifies problems so that they can be fixed before they affect system performance. Furthermore, fixing such problems also requires accurate information about system coverage over time, on a per-component level, so that grammar writers can effectively track down and correct any loss of coverage, as well as identify areas for further development.

Here we focus on two closely related regression systems used in grammar-based systems: one developed at Geneva University for a multi-lingual medical spoken translation system based on the Regulus platform (Rayner et al., 2006), and one for a search and question-answering system that uses the XLE LFG parser and ordered-rewriting system (Crouch et al., 2007). Given the usefulness of these tools for two disparate systems at whose cores are heavily engineered deep grammars, we hope that the techniques described here will be useful to the grammar engineering community regardless of framework, especially since these grammars are increasingly used as central system components. Although many aspects of the regression testing we describe here can be attributed to common sense, the paper pulls together lessons learned in the development and active use of the systems. This includes the fact that ease of use and a well-designed user interface are of great help, even for experienced system developers.

Regression testing for grammar-based systems involves two phases. The first includes systematic testing of the grammar during its 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 grammar 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 test suites so that the developers do not need to change the gold standard annotation every time an intermediate level of representation changes.

In the following we first describe in more detail how the regression tests are used in grammar development (Section 2), and then how they are used for the grammars within a larger system and application environment (Section 3), drawing examples from two systems using complex grammars.

2 Regression Testing During Grammar Development

Most large-scale grammar development efforts use regression testing for the output of their grammars. Many complex systems have separate tests for different levels of output, e.g. syntactic and semantic. The test suites are often defined by the grammar writer, perhaps in conjunction with some corpus, and they aim for coverage of (specific) linguistic and lexical data (see Lehmann et al. (1996) on test suite design for NLP). Here we do not explore the development of these test suites, including tools to assist in the creation of appropriate "gold" standard analyses to compare against (Oepen et al., 1998; Rosén et al., 2005). Instead, we focus on the regression system a grammar developer would need to maximally benefit from the test suites that they have.

The basic idea is extremely simple. Having constructed the regression corpora with annotations as to what the correct outputs are, the system must run the grammar against each test suite and notify the developers how things have changed. The details of how the regression testing system is run and how results are presented are crucial. If the regression testing tools are badly designed, the testing will be time consuming and laborious to the point that it is not performed on a regular basis. If the details are right, it makes a huge difference in ease of grammar development and in overall application maintenance. As we will see, these feature requirements are similar to those of regression testing of the system as a whole.

In addition to the ability to run the tests manually after each change, the regression system should run automatically on a regular schedule, generally overnight so that the results are ready for the developer to review each morning. These results need to be posted in such a way that they can easily be perused by the grammar writer. For example, a result summary might be mailed to the developer, along with a link to a web page with detailed results of the regression run.

A second important feature is a method to compare the results of different test runs. Most regression sets do not yield perfect results, so the grammar developer needs some way to determine whether the incorrect results indicate something that was recently broken, as opposed, for example, to something that has not been implemented yet. Furthermore, when an example gives a wrong result, it is necessary for the regression system to report if the grammar ever gave the right result, what it was, and when that result was last produced. Hence it is necessary to have tools that enable the grammar engineer to compare the latest results against the immediately previous ones, as well as against the best ever result and the gold standard result for a given example. Examples of displays are shown in the next section.

3 Regression Testing the Grammar in the System

As anyone who has worked as a grammar engineer is painfully aware, linguistic formats change over time. Sometimes these are minor changes, such as the systematic renaming of features, while other changes may involve significant linguistic reanalysis. Annotation schemes that rely on internal representations (parse trees, semantic forms, etc.) run into the problem that the annotations themselves degrade as the representations change. As such, it is not easy to know whether mismatches occurring during regression runs reflect a real problem or just a case of a change in internal representation.[1] For this reason, it is good to have test suites whose annotations do not refer to internal forms; these suites can supplement the more traditional grammar-based test suites discussed briefly in the previous section.

For example, in a dialog system, the annotation can state whether or not Y is a good response to X in a given context. This scheme has been implemented in the Clarissa dialog manager (Rayner and Hockey, 2004). In the Clarissa system, regression testing is performed by performing dialog moves on input states, producing a specific output state and a set of required actions. A context-independent test suite can be built by recording the desired output state and accompanying actions for corresponding input states. In a translation system, annotations can state whether or not a Y is a good translation of X, instead of testing the syntactic representations used internally to perform the translation. In a question answering system, they can state whether passage X should match query Y. In most applications, there can be multiple correct responses, e.g. an MT system can produce multiple good translations. As such, the regression testing must allow for this possibility (see section 3.1.2). These considerations lead us to the conclusion that it is often easiest to perform regression testing of a grammar in the context of a larger system that uses the grammar to perform some concrete task (Spark-Jones and Galliers, 1996).

In this section we describe the Regulus-based medical speech translation system MedSLT and the XLE-based search and question answering systems.

[1]Several annotation systems focus on this problem, providing tools, such as discriminant features, that can be used to quickly bootstrap a new regression suite off of a previous version (Oepen et al., 2002; Rosén et al., 2005).

3.1 Regression Testing Regulus Grammars in MedSLT

3.1.1 The Regulus Toolkit

Regulus is a comprehensive Open Source toolkit for developing grammar-based speech-enabled systems that can be run on the commercially available Nuance speech recognition environment. The platform has been developed by an Open Source consortium, whose main partners have been NASA Ames Research Center (ARC) and Geneva University, and is freely available for download from the project's SourceForge website. The platform supplies a framework which supports development of grammars, compilation of grammars into parsers, generators and recognisers, and use of these compiled resources to build speech translation and spoken dialog applications. It has been used to build several large systems, including NASA's Clarissa procedure browser, and is described at length in Rayner et al. (2006).

The Regulus development model is based on reusable grammars. Developers use general domain-independent unification grammars, which are tailored to the domain at hand by *grammar specialization* using explanation-based learning (EBL). Grammar specialization is conducted using a small training corpus, domain specific lexica, and a set of instructions ("operationality criteria") describing how to perform the specialization (see Figure 1).

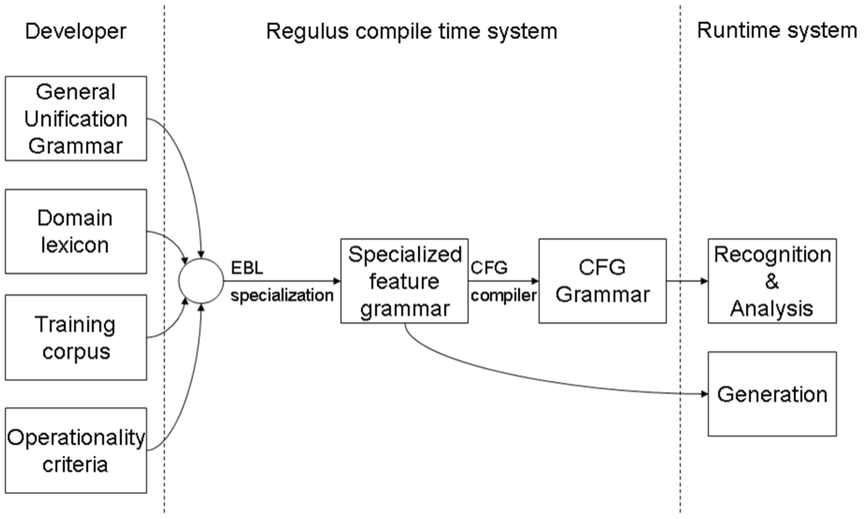
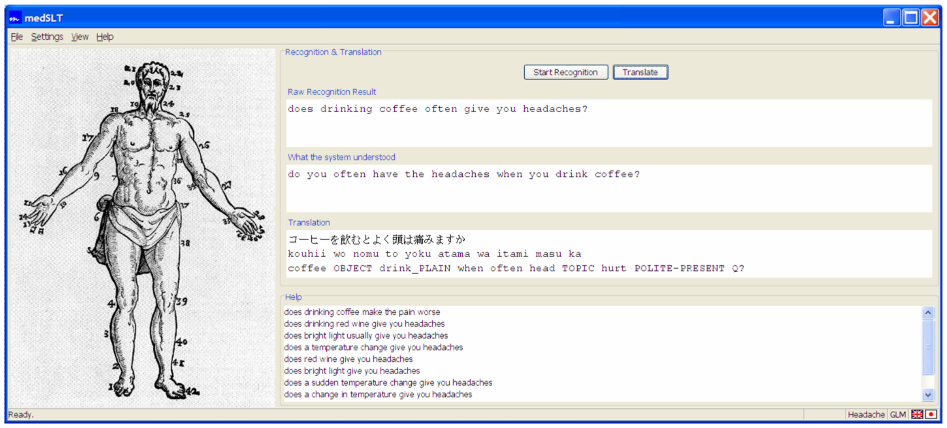


Figure 1: The Regulus processing path

3.1.2 The MedSLT System

Geneva University's MedSLT is a large Regulus-based project, which focuses on automatic interactive translation of spoken doctor/patient examination dialogs (for a screenshot, see Figure 2).



```
does drinking coffee often give you headaches?
```

```
do you often have the headaches when you drink coffee?
```

```
コーヒーを飲むとよく頭は痛みますか
kouhii wo nomu to yoku atama wa itami masu ka
coffee OBJECT drink_PLAIN when often head TOPIC hurt POLITE-PRESENT Q?
```

does drinking coffee make the pain worse
does drinking red wine give you headaches
does bright light usually give you headaches
does a temperature change give you headaches
does red wine give you headaches
does bright light give you headaches
does a sudden temperature change give you headaches
does a change in temperature give you headaches

Figure 2: MedSLT application window

As of 2007, there are versions of the system for approximately twenty different language pairs and four subdomains. The subdomains covered are headaches, chest pain, abdominal pain and pharyngitis. Languages handled include English, French, Japanese, Spanish, Catalan and Arabic. Vocabulary size varies depending on the input language, being determined mainly by the number of inflected forms of verbs and adjectives. It ranges from ∼400 for Japanese to ∼1100 for French. The overall system is stable, and has been tested on medical students in simulated patient-examination situations with good results (Chatzichrisafis et al., 2006).

Typically the Regulus grammar for each MedSLT language is used for several system components, including speech recognition, analysis, and generation of translated sentences. Maintenance and active extension of these different components for the many different language-pair and domain combinations led the MedSLT developers to build an elaborate set of regression testing tools. Each language and domain have regression corpora in text form. Additionally each language has recorded speech corpora for selected domains. The regression testing produces results at four different levels: individual utterances, individual corpus runs, sets of runs for a language, and the complete set.

During regression testing, each utterance is passed through all stages of processing in order to determine how the system as a whole is performing. For MedSLT, a spoken dialog translation system, speech is an important aspect, and is thus thoroughly tested using offline speech recognition with the Nuance `batchrec` utility. Automated tests provide the developers objective runtime performance figures, and enable the team to tune grammars and recognition parameters to match the target platform. Speech recognition regression tests report for each test suite semantic error rates, word error rates, and run-time performance of the test suite in terms of CPU time.

The next processing steps cover source language analysis, ellipsis resolution, translation to interlingua, translation from interlingua, and target language genera-

```
Wavfile: c:/corpora/2004-11-01/USEnglish/13:34:41/utt19.wav
Source: does your pain appear in the morning
                +avez-vous mal des deux cotes
Recognised: does your pain appear in the morning
Target: la douleur survient-elle le matin
*** PREVIOUS OK: "avez-vous mal le matin" ***
 Source rep:   [[possessive,[[[pronoun,you]]]],
                [prep,in_time], [secondary_symptom,pain],
                [state,appear], [tense,present],
                [time,morning], [utterance_type,ynq],
                [voice,active]]
 Resolved rep: [[prep,in_time], [symptom,pain],
                [pronoun,you], [state,have_symptom],
                [tense,present], [time,morning],
                [utterance_type,ynq], [voice,active]]
   Resolution: trivial
  Interlingua: [[prep,in_time], [pronoun,you],
                [state,have_symptom], [symptom,pain],
                [tense,present], [time,morning],
                [utterance_type,ynq], [voice,active]]
   Target rep: [[state,survenir], [symptom,douleur],
                [temporal,matin], [tense,present],
                [utterance_type,sentence], [voice,active]]
     Judgment: ?
```

Figure 3: Result record for individual utterance (slightly simplified). The lines show, in order: the name of the recorded speech file; a transcription of the source utterance with preceding context; the recognised result; the translation; a previously produced correct translation; various internal representations; and the quality judgment (currently unknown '?').

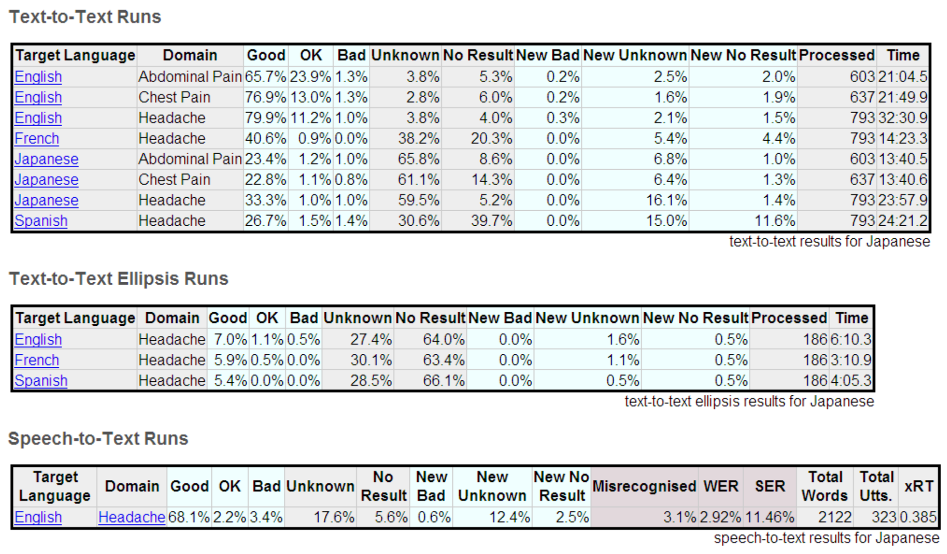
tion. Performance on these steps is summarized by showing the quality of the translated corpus and the number of sentences that did not produce a result. To summarize translation quality of the corpus, translations are judged as being Good, OK (acceptable but not perfect) and Bad.[2] A database of all previous results is stored, so that Bad results can show when the example most recently produced a correct result and what that correct result was. To help keep judgments up to date, the regression testing scripts rebuild the judgment databases, and any sentences without a matching judgment are flagged with '?'. Developers are able to click on the corresponding corpus and start annotating these flagged sentence pairs as Good, OK or Bad.

[2]Judgments are based on both syntactic and semantic criteria. Sentences with good syntax and semantics are judged as Good. In case the semantics is correct, but the syntax could be improved, the sentence would be categorized as OK, otherwise as Bad.

A typical result record for an utterance is shown in Figure 3. In this particular run, a new translation was produced (*la douleur survient-elle le matin*) presumably due to a change in the translation rules, and no stored judgment was available. The system flags this ('?') and shows the user a known OK translation previously produced (*avez-vous mal le matin*).

3.1.3 How the Regression Testing Tools Drive the Debugging Process

This section presents an illustrative example, showing how the regression testing tools are used in the normal MedSLT development cycle. The grammar engineer starts by examining the top-level webpage for the nightly corpus run. This displays a formatted table, with one line for each corpus; tables for individual languages are grouped on different tabs. Figure 4 shows the set of tables for Japanese input. Each line summarizes the judged quality of translations (Good, OK, Bad), how many translation have not been judged yet (Unknown), and for how many sentences no translation is produced (NoResult). Furthermore, the columns NewBad, NewUnknown, and NewNoResult show how the results have changed from the previous run. The number of processed sentences (Processed) and time used (Time) are printed in the last two columns.



Text-to-Text Runs

Target Language	Domain	Good	OK	Bad	Unknown	No Result	New Bad	New Unknown	New No Result	Processed	Time
English	Abdominal Pain	65.7%	23.9%	1.3%	3.8%	5.3%	0.2%	2.5%	2.0%	603	21:04.5
English	Chest Pain	76.9%	13.0%	1.3%	2.8%	6.0%	0.2%	1.6%	1.9%	637	21:49.9
English	Headache	79.9%	11.2%	1.0%	3.8%	4.0%	0.3%	2.1%	1.5%	793	32:30.9
French	Headache	40.6%	0.9%	0.0%	38.2%	20.3%	0.0%	5.4%	4.4%	793	14:23.3
Japanese	Abdominal Pain	23.4%	1.2%	1.0%	65.8%	8.6%	0.0%	6.8%	1.0%	603	13:40.5
Japanese	Chest Pain	22.8%	1.1%	0.8%	61.1%	14.3%	0.0%	6.4%	1.3%	637	13:40.6
Japanese	Headache	33.3%	1.0%	1.0%	59.5%	5.2%	0.0%	16.1%	1.4%	793	23:57.9
Spanish	Headache	26.7%	1.5%	1.4%	30.6%	39.7%	0.0%	15.0%	11.6%	793	24:21.2

text-to-text results for Japanese

Text-to-Text Ellipsis Runs

Target Language	Domain	Good	OK	Bad	Unknown	No Result	New Bad	New Unknown	New No Result	Processed	Time
English	Headache	7.0%	1.1%	0.5%	27.4%	64.0%	0.0%	1.6%	0.5%	186	6:10.3
French	Headache	5.9%	0.5%	0.0%	30.1%	63.4%	0.0%	1.1%	0.5%	186	3:10.9
Spanish	Headache	5.4%	0.0%	0.0%	28.5%	66.1%	0.0%	0.5%	0.5%	186	4:05.3

text-to-text ellipsis results for Japanese

Speech-to-Text Runs

Target Language	Domain	Good	OK	Bad	Unknown	No Result	New Bad	New Unknown	New No Result	Misrecognised	WER	SER	Total Words	Total Utts.	xRT
English	Headache	68.1%	2.2%	3.4%	17.6%	5.6%	0.6%	12.4%	2.5%	3.1%	2.92%	11.46%	2122	323	0.385

speech-to-text results for Japanese

Figure 4: Part of display summarising the results of a nightly MedSLT test run. Ellipsis processing for Japanese is not yet well developed; speech recognition is in contrast very good.

Looking at the tables it is immediately apparent that performance on Japanese to Spanish has slipped badly (11.6% in "New No Result"). On the other hand, although performance on the ellipsis corpus is not good, little appears to have broken; the poor result is due to the fact that most of the Japanese ellipsis processing rules have not yet been implemented. The developer's next step is thus to examine the detailed trace for the main Japanese to Spanish corpus.

Having clicked to get the corpus trace, the developer now searches down the file for occurrences of the string "PREVIOUS". As mentioned above, examples which used to work and now give different results are flagged in this way. In the present example, a minute or so of searching is enough to show that most or all examples involving the common Japanese word *tsuzuku* 'continue/last' are failing to produce translations; a target language representation is created, but surface generation fails, as in the example shown in Figure 5. This is a detailed enough analysis to permit direct debugging of the problem in the Regulus development environment.

```
Source: issyuukan ijou tsuzuki masu ka
Target: generation_failed
*** NO TRANSLATION ***
*** PREVIOUS OK: "el dolor dura mas de una semana" ***
    Source rep: [[number,1], [numerical_comparative,ijou],
                 [state,tsuzuku], [tense,present],
                 [unit,syuukan], [utterance_type,sentence]]
  Resolved rep: [[utterance_type,sentence], [number,1],
                 [unit,syuukan], [numerical_comparative,
                 ijou], [tense,present], [state,tsuzuku]]
    Resolution: trivial
   Interlingua: [[prep,duration], [spec,[more_than,1]],
                 [state,last], [symptom,pain],
                 [tense,present], [timeunit,week],
                 [utterance_type,ynq], [voice,active]]
    Target rep: [[comparative,mas_de], [number,1],
                 [state,durar], [symptom,dolor],
                 [tense,present], [timeunit,semana],
                 [utterance_type,sentence], [voice,active]]
      Judgment: error
```

Figure 5: Result record for an individual utterance (slightly simplified) showing an example of a Japanese to Spanish translation problem. The input sentence could be glossed as "One-week more continue POLITE Q" ("Has [the pain] continued for more than one week?"). The meanings of the other fields are described in the caption to Figure 3.

In general, it is extremely useful to be able to descend rapidly in this way from the top-level display, which shows figures for over twenty different language pairs, to the low-level task of debugging a single representative test suite example. The point to note here is that good organization of the trace information has allowed the language engineer to be fairly sure, after only a few minutes, that this is indeed one of the most important outstanding problems. Having a reliable testing framework of this kind makes it possible to focus developer effort effectively, and facilitates overall project management.

3.2 Regression Testing in the XLE QA System

The goal of the XLE search and question answering (QA) system (Bobrow et al., 2007; Crouch and King, 2006) is to find matches between passages and queries, where a “match” is defined, depending on the application, as anything from strict entailment (strict QA) to relevancy (search). The system batch processes texts, mapping them into deep semantic representations, and stores these in a semantically-indexed database. For retrieval or question answering, the query is mapped through a related set of rules, and the query representation is used to retrieve relevant passages from the index. These are then ranked as to relevance (search) or run through a set of entailment and contradiction detection rules (question answering).

In order to run regression testing on this system, passage and query pairs with answers are created and run through the syntax to the semantic representations. A light inference procedure is applied to detect entailments or contradictions between the final representations of passages and queries. The resulting answer is compared against the gold standard answer.[3] The basic system architecture as used by the regression testing system is shown in Figure 6.

$$\begin{tikzcd}
 & \text{input string} \arrow[d, "\text{Finite-State Machines}"] & \\
 & \text{textbreaking tokenization} \arrow[d, "\text{Finite-State Machines}"] & \\
 & \text{morphological analysis} \arrow[d, "\text{XLE LFG parser}"] & \\
 & \text{syntactic analysis} \arrow[d, "\text{XLE rewrite system}"] & \\
 & \text{semantics} \arrow[dl] \arrow[dr, "\text{XLE rewrite system}"] & \\
\text{query reformulation} \arrow[dr] & & \text{passage expansion} \arrow[dl, "\text{XLE rewrite system}"'] \\
 & \text{match inference} \arrow[d] & \\
 & \text{answer} &
\end{tikzcd}$$

Figure 6: XLE Search and QA System Processing Pipeline

The regression system is set up to provide feedback to the rule writers maintaining both the semantic production system and the inference mechanism. In contrast to the MedSLT application, the XLE Search and QA System only involves one language. Nonetheless, there are several grammar engineers working on the different semantics levels, the inference rules, and the lexical resources used by the system. In addition, the tokenization, morphology, and syntax preprocessing (Kaplan et al.,

[3]This regression platform does not currently test the indexing mechanism. This capability is in the process of being added.

2004) slowly change over time and updates to these components can affect downstream processing in terms of the quality of the answers and of system efficiency.

3.2.1 Regression Testing Question Answering

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).[4] Given an application of use, it is rarely ambiguous what the answer should be. In contrast, one input to a translation system can have many possible outputs of varying correctness that are hard to enumerate. The upshot for QA systems is that regression runs are simpler and easier to interpret once the test suites have been constructed.

Regression test suites in the XLE QA system are separated into three groups: sanity sets, phenomenon sets, and real-world sets.

Sanity sets The entailment and contradiction detection part of the system is tested in isolation by matching queries against themselves (e.g. a passage *John walks.* is tested against a query *John walks.*); note that queries in this system do not have to be syntactically interrogative. The sanity check test suites 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. These tests are run nightly.

Phenomena sets Real-world sentences require analyses of multiple interacting phenomena. Naturally, longer sentences tend to have more diverse sets of phenomena and hence a higher chance of containing a construction that the system does not deal with well. This can lead to frustration for system engineers trying to track progress; fixing a major piece of the system can have little or no effect on a small sample of real-world examples. To alleviate this frustration we have sets of hand-crafted test examples that are focused as much as possible on single phenomena, e.g. anaphora, aliases, deverbals, implicatives. These include externally developed test suites such as the FraCaS (Cooper et al., 1996) and HP test suites (Nerbonne et al., 1988). These focused test suites are also good for quickly diagnosing problems. If all broken examples are in the deverbal test set, for example, it gives system engineers a good idea of where to start looking for bugs. These are the most important tests and are run nightly.

[4] *Wh*-questions receive a yes answer if an alignment is found between the *wh*-word in the query and an appropriate part of the 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. However, currently not all of the test suites include gold standards for this alignment.

Real-world sets The ultimate goal of the system is to work on real-world examples; so tests of those are important for assessing progress on naturally occurring data. These test suites are created by extracting sentences from corpora expected to be used in the run-time system, e.g. newspaper text or the Wikipedia. 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 used to ensure that the real-world sets use an appropriate range of queries. Currently, the XLE systems use a combination of hand-crafted queries and queries from the RTE data which were hand-crafted, but not by the XLE QA system developers. These tests are run once a week.

3.2.2 Comparing with Previous Regression Results

The point of regression testing is to compare system outputs over time. The XLE regression system automates this task as much as possible. Performance on individual test suites is graphed over time. Each test suite can be viewed individually in this way. An example is seen in Figure 7 in which a sharp dip in performance is seen for August 19 but coverage climbs after that, with a plateau in early September.

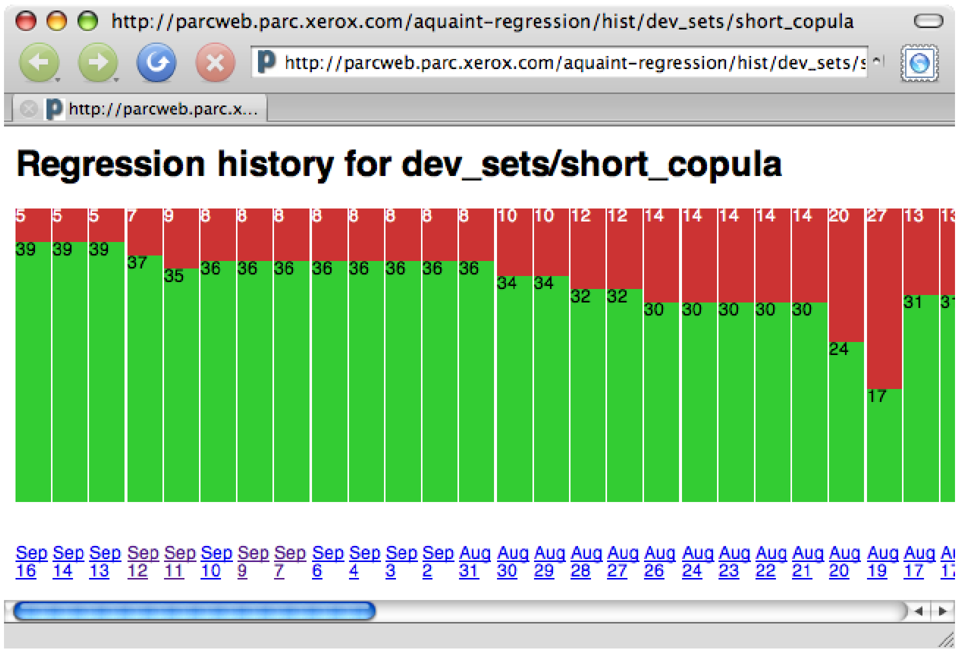
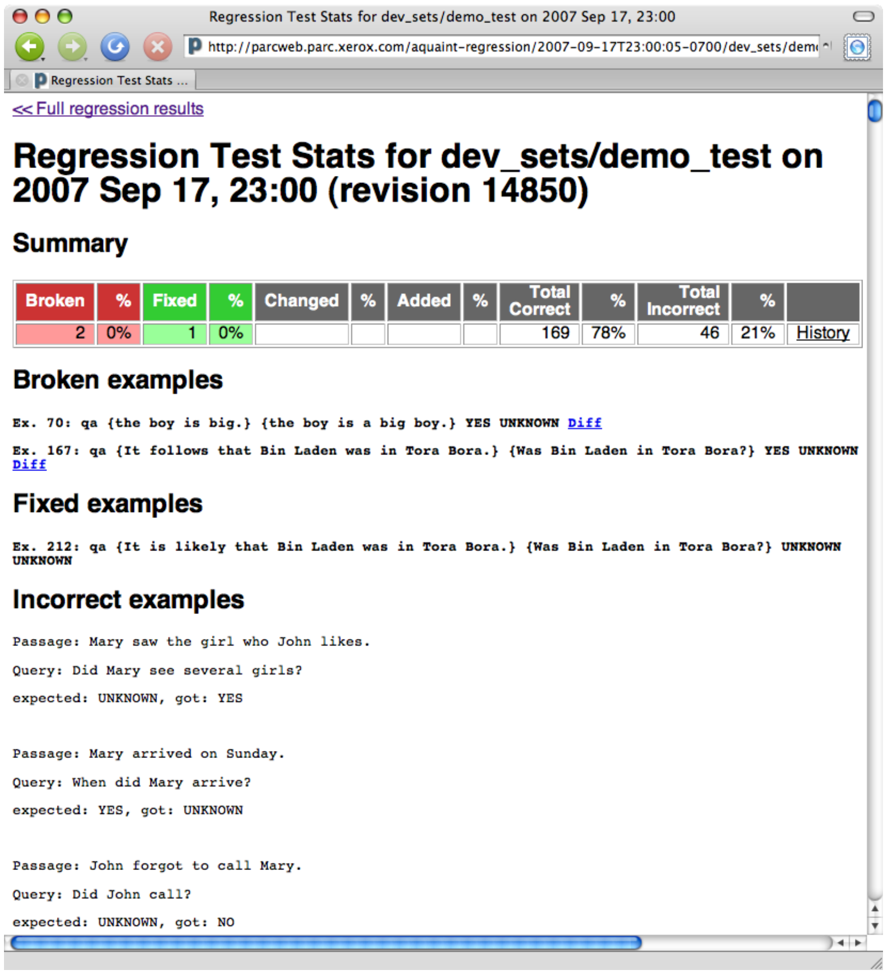


Figure 7: XLE Regression System: Performance over time

In addition, a view of each test suite is available in which the broken and fixed examples are placed at the top of the page, directly under the result summary for that test suite. This is shown in Figure 8. Below those, all the incorrect pairs are shown, then the correct ones, then a full system log.

Diffs of output, including debugging information and representations, between different versions of the system can be easily generated. The query-passage pair is


Regression Test Stats for dev_sets/demo_test on 2007 Sep 17, 23:00
http://parcweb.parc.xerox.com/aquaint-regression/2007-09-17T23:00:05-0700/dev_sets/dem
Regression Test Stats ...

<< Full regression results

Regression Test Stats for dev_sets/demo_test on 2007 Sep 17, 23:00 (revision 14850)

Summary

Broken	%	Fixed	%	Changed	%	Added	%	Total Correct	%	Total Incorrect	%	
2	0%	1	0%					169	78%	46	21%	History

Broken examples

```
Ex. 70: qa {the boy is big.} {the boy is a big boy.} YES UNKNOWN Diff
Ex. 167: qa {It follows that Bin Laden was in Tora Bora.} {Was Bin Laden in Tora Bora?} YES UNKNOWN
Diff
```

Fixed examples

```
Ex. 212: qa {It is likely that Bin Laden was in Tora Bora.} {Was Bin Laden in Tora Bora?} UNKNOWN
UNKNOWN
```

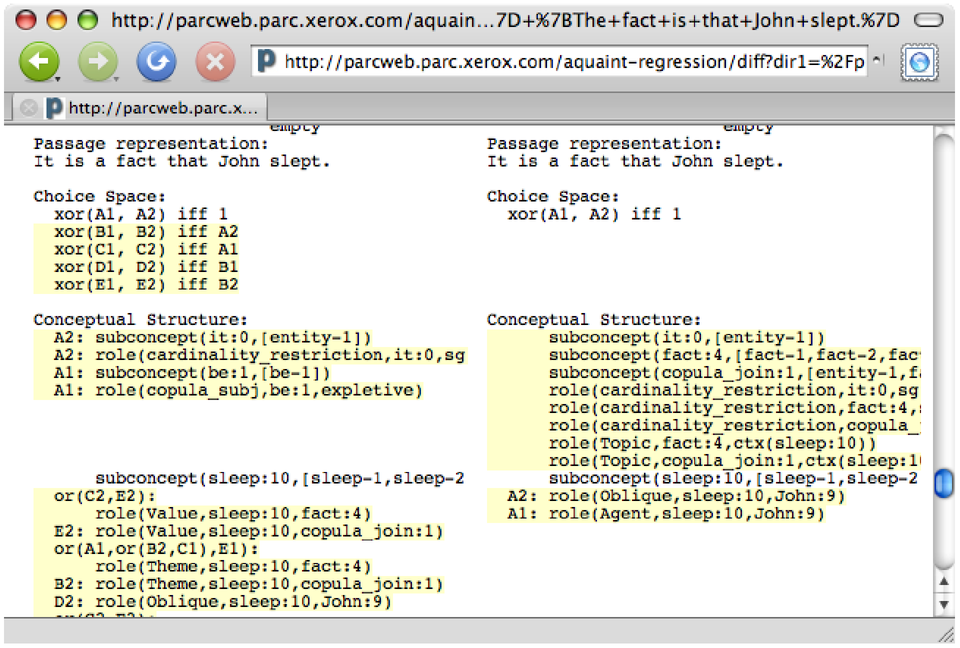
Incorrect examples

```
Passage: Mary saw the girl who John likes.
Query: Did Mary see several girls?
expected: UNKNOWN, got: YES

Passage: Mary arrived on Sunday.
Query: When did Mary arrive?
expected: YES, got: UNKNOWN

Passage: John forgot to call Mary.
Query: Did John call?
expected: UNKNOWN, got: NO
```

Figure 8: XLE Regression System: Detailed presentation of test suite results

run with light-weight debugging on in two versions of the system and resulting differences in the rule application and the representations are highlighted. This makes it easy for the developers to see the types of representations that were produced by previous versions of the system in comparison with the current version. In addition, the diffs of the rules triggered by each run allow the developer to see more precisely where any divergences occur. The most frequently used diff is between the current system and the previous day's. Part of a sample diff is shown in Figure 9 in which the previous day's run, shown on the left, has more possible analyses, as indicated by the larger choice space. However, images of previous days' systems are stored for rapid comparison, and for comparisons further back in time, system versions are retrieved from the svn repository.

Each time the regression testing completes a run (nightly for most test suites), an email message is sent to the developers with a summary of who committed changes



```
Passage representation:                       Passage representation:
It is a fact that John slept.                 It is a fact that John slept.

Choice Space:                                 Choice Space:
  xor(A1, A2) iff 1                             xor(A1, A2) iff 1
  xor(B1, B2) iff A2
  xor(C1, C2) iff A1
  xor(D1, D2) iff B1
  xor(E1, E2) iff B2

Conceptual Structure:                         Conceptual Structure:
  A2: subconcept(it:0,[entity-1])                   subconcept(it:0,[entity-1])
  A2: role(cardinality_restriction,it:0,sg          subconcept(fact:4,[fact-1,fact-2,fac
  A1: subconcept(be:1,[be-1])                       subconcept(copula_join:1,[entity-1,f
  A1: role(copula_subj,be:1,expletive)              role(cardinality_restriction,it:0,sg
                                                    role(cardinality_restriction,fact:4,
                                                    role(cardinality_restriction,copula_
                                                    role(Topic,fact:4,ctx(sleep:10))
                                                    role(Topic,copula_join:1,ctx(sleep:1
      subconcept(sleep:10,[sleep-1,sleep-2          subconcept(sleep:10,[sleep-1,sleep-2
  or(C2,E2):                                    A2: role(Oblique,sleep:10,John:9)
      role(Value,sleep:10,fact:4)               A1: role(Agent,sleep:10,John:9)
  E2: role(Value,sleep:10,copula_join:1)
  or(A1,or(B2,C1),E1):
      role(Theme,sleep:10,fact:4)
  B2: role(Theme,sleep:10,copula_join:1)
  D2: role(Oblique,sleep:10,John:9)
```

Figure 9: XLE Regression System: Sample Diff for the sentence *It is a fact that John slept.*

and how many examples were fixed and broken. In addition, a link is provided to a web server with the current graphs and diffs, as well as with links to previous results. The combination of automatic nightly regression runs with the graphical presentation of results has proven vital for the maintenance and development of the QA system.

4 Conclusions

As deep grammars are increasingly used as components of larger systems, reliable, accurate, and easy-to-use regression testing is crucial. Here, we have described regression testing techniques used to maintain large-scale grammars in two applications. The regression testing runs automatically each night and reports to the grammar developers how the performance of the system has changed. The reports include summary information as well as which examples changed, how they changed, and when they last worked correctly. It is our hope that other grammar engineering efforts can benefit from our experiences in order to more rapidly and effectively maintain and develop their grammars for applications.
]

References

Bobrow, Danny, Cheslow, Bob, Condoravdi, Cleo, Karttunen, Lauri, King, Tracy Holloway, Nairn, Rowan, de Paiva, Valeria and Zaenen, Annie. 2007. Linguistically-based Textual Inference. In Tracy Holloway King and Emily M. Bender (eds.), *Proceedings of the GEAF 2007 Workshop*, CSLI Publications.

Chatzichrisafis, Nikos, Bouillon, Pierrette, Rayner, Manny, Santaholma, Marianne, Starlander, Marianne and Hockey, Beth Ann. 2006. Evaluating Task Performance for a Unidirectional Controlled Language Medical Speech Translation System. In *Proceedings of the HLT-NAACL Workshop on Medical Speech Translation*.

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

Crouch, Dick, Dalrymple, Mary, Kaplan, Ron, King, Tracy Holloway, Maxwell, John and Newman, Paula. 2007. XLE Documentation, on-line documentation.

Crouch, Dick and King, Tracy Holloway. 2006. Semantics via F-Structure Rewriting. In *Proceedings of LFG06*, pages 145–165, CSLI Publications.

Kaplan, Ron, Riezler, Stefan, King, Tracy Holloway, Maxwell, John T., Vasserman, Alex and Crouch, Richard. 2004. Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of HLT-NAACL'04*.

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

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

Oepen, Stephan, Flickinger, Dan, Toutanova, Kristina and Manning, Chris D. 2002. LinGO Redwoods. A Rich and Dynamic Treebank for HPSG. In *Proceedings of The First Workshop on Treebanks and Linguistic Theories*.

Oepen, Stephan, Netter, Klaus and Klein, Judith. 1998. TSNLP — Test Suites for Natural Language Processing. In John Nerbonne (ed.), *Linguistic Databases*, CSLI.

Rayner, Manny and Hockey, Beth Ann. 2004. Side Effect Free Dialogue Management in a Voice Enabled Procedure Browser. In *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP)*, Jeju Island, Korea.

Rayner, Manny, Hockey, Beth Ann and Bouillon, Pierrette. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI.

Rosén, Victoria, de Smedt, Koenraad, Dyvik, Helge and Meurer, Paul. 2005. TREPIL: Developing Methods and Tools for Multilevel Treebank Construction. In *Proceedings of The Fourth Workshop on Treebanks and Linguistic Theories.*

Spark-Jones, Karen and Galliers, Julia Rose. 1996. *Evaluating Natural Language Processing Systems*. Springer Verlag.