The Role of Testing in Grammar Engineering

Martin Volk
University of Koblenz-Landau
Institute of Computational Linguistics
Rheinau 3-4
5400 Koblenz, Germany
(+49)-261-9119-469
e-mail volk@brian.uni-koblenz.de

1 Introduction

In the past grammars have been developed either with an art approach (building up a coherent system; prescriptive grammars like the Latin grammars of the Middle Ages) or with a science approach (describing the laws of nature; descriptive and contrastive grammars). We propose to regard grammar development in Computational Linguistics as an engineering task analogous to software engineering: one that requires analysis, specification, implementation, testing, integration and maintenance.

The different phases in the software development process correspond to phases in grammar development in the following way:

- The problem analysis and definition phase corresponds to an analysis of the linguistic data (texts of written or spoken language).
- Problem specification means setting up grammar rules that describe the observed data. The grammar formalism thus provides the formal language for the specification.
- The implementation phase includes putting the rules into the specific format of the computational grammar system. This is a computer program to process a grammar in the framework of a grammar theory. The implementation effort decreases the closer the format of the grammar approaches the format of the computational system.
- The testing phase comprises checking the grammar implementation against the linguistic data, i.e. judging grammaticality and the assigned structure.
- Integration, installation and maintenance are no different in the context of an NLP system than in other software systems.

Our project focusses on the testing aspect of the process. Testing can never be exhaustive but must be representative. We therefore propose an archive of test sentences (hence: ATS) to govern the incremental development and the testing of grammar implementations.

2 Construction of an ATS

Towards the goal of a comprehensive collection of test sentences we have restricted ourselves to the construction of an ATS that represents specific *syntactic* phenomena. The archive aims to be a representative sample of all syntactic phenomena of a natural language, in our case German.

The ATS must contain grammatical sentences as well as ungrammatical strings. The grammatical sentences are systematically collected by varying one syntactic phenomenon at a time. E.g. we vary first the subcategory of the verb and then we vary verb tense, etc. For every phenomenon we have to construct ungrammatical strings to check against overgeneration by the grammar. These strings are found by manipulating the phenomenon in question in such ways as to make it ungrammatical. A syntactic feature must be varied over all values of its domain. E.g. the feature *case* in German has the values *nominative*, *genitive*, *dative*, and *accusative*. A noun phrase (NP) that needs to appear in the nominative in a given sentence will then result in three ungrammatical strings when the other cases are assigned.

It must be noted that the set of ungrammatical strings gets very large when it comes to problems such as word order where the permutation of all the words in a sentence is necessary to enumerate all possibilities. In this case we need heuristics to find the most appropriate test sentences. E.g. in German the order of NPs in a sentence is variable with certain restrictions whereas the word order within the NP is relatively fixed. Therefore we are much more likely to encounter problems in NP order when setting up our grammar. As a result we will have to focus on ungrammatical strings with NP order problems. In contrast to grammatical sentences ungrammatical strings are only seldom found naturally (e.g. errors of language learners) and it will be interesting to study whether these occurrences (at least the most frequent ones) correspond to our constructed examples.

The judgement of grammatical versus ungrammatical strings is subjective and has little to say about the acceptability of a sentence in a real-world communication. Thus our ATS will model competence rather than performance in the Chomskyan sense.

For the practical use the ATS must be organized in a modular fashion, enabling the user to adapt and extend the archive according to his needs and convictions. Furthermore it must be documented why a sentence has been entered into the archive, since every sentence displays a multitude of syntactic information, only some of which is relevant in our domain.

3 Testing with an ATS

The ATS can be useful in many respects. First, it can govern the development process of a grammar in that it provides the linguistic data in a concise fashion. Grammar rules can be incrementally constructed to account for the given sentences. Organizing the ATS around syntactic phenomena of increasing complexity supports incremental grammar development. After each phenomenon has been formalized, the grammar can be checked against the test sentences, thus facilitating the retrieval of problematical sections in the grammar. The goal is to set up the archive in such a way that incremental grammar development does not require total retesting of all the previous material but only of the recent relevant phenomena. But foremost the ATS is meant to support the testing of the grammar for completeness (all sentences that should be accepted are accepted by the grammar) and soundness (the grammar does not accept any sentence that should not be accepted).

In testing a grammar we need to distinguish between using the grammar for analysis or synthesis. In analysis the ATS can be used to provide input sentences for the parsing process. In synthesis the ATS can be used for comparison with the generated sentences and thus minimize the human judgement effort to the leftover sentences. The grammatical ones of this group can be used to complete the ATS.

We see three major advantages in using an ATS.

1. Organizing the ATS in equivalence classes around syntactic phenomena facilitates incremental grammar development and problem driven testing.
2. Compared to working with NL corpora the ATS avoids redundant testing of frequently occuring syntactic phenomena. The ATS can thus serve as a testbed before the grammar is used on real ("unconstructed") texts.
3. The ATS will help to compare the performance of different implementations within the same formalism as well as across linguistic theories. Running an LFG implementation against a GPSG implementation will show the respective adequacy of the theories for particular syntactic phenomena.

In order to apply an ATS in practice we have built a workbench for experimentation with grammar development that contains an archive of this nature (see Volk and Ridder, 1991). The workbench is designed as a tutorial system for students of syntax analysis in Computational Linguistics. Its ATS contains 350 sentences for 15 syntactic phenomena. The workbench comes with a lexicon, a morphology component, a special purpose editor and output procedures for linguistic type tree output. The program is written in Prolog and runs on PCs.

4 Similar work

Concerning the ATS the approach most similar to our own is by Nerbonne and coworkers (1991). They assemble a database of constructed sentences in an attempt to exhaustively cover syntactic phenomena. So far they have tackled coordination and verb government with several hundred sentences for both. They have not yet included their "diagnostic database" in any test system. This work also reports on attempts to build up sentence collections for English.

Other comparable approaches towards grammar engineering are by Erbach (1991) and by Erbach and Arens (1991). The first describes a system that allows for the parametrization of and experimentation with different parsing strategies. The user can specify priorities to incrementally optimize the parsing process. We believe, however, that lacking a broad collection of test sentences this system cannot be sufficiently evaluated and therefore we see our approach as complementary to theirs.

In another attempt Erbach and Arens (1991) try to evaluate grammars by generating a "representative set of sentences" in a systematic way. They limit their lexicon and grammar, and starting with sentences of length 1, they generate sentences with increasing length. It is not clear how they intend to check the resulting sentences other than by human judgement. An ATS that is adapted to their lexicon could be compared against these sentences.

5 Future directions

Future work will focus on two aspects. First, we will try to apply testing techniques from software engineering to our domain of grammar development. In particular we hope to demonstrate that building a grammar implementation is a special case of declarative programming, since most recent grammar formalisms (notably the unification-based theories) are declarative in nature. This needs to go together with test statistics and precise information on how to incrementally improve the grammar.

Second, in the long run it will be necessary to add sentences that exceed pure syntactic testing and check for semantic regularities. It is by no means clear how the test sentences for semantics should be collected since there is no set of agreed upon semantic features that could be varied.

References

Gregor Erbach. *An Environment for Experimentation with Parsing Strategies.* (IWBS Report 167) Stuttgart: Wissenschaftliches Zentrum der IBM Deutschland. April 1991.

Gregor Erbach and Roman Arens. Evaluation von Grammatiken für die Analyse natürlicher Sprache durch Generierung einer repräsentativen Satzmenge. *Proceedings of GWAI-91*, pages 126-129, Bonn, September 1991.

John Nerbonne et al. *A diagnostic tool for German syntax.* (Research Report RR-91-18) Saarbrücken: DFKI. July 1991.

Martin Volk and Hanno Ridder. *GTU (Grammatik Test Umgebung) Manual.* (Manuscript) Institute of Computational Linguistics. University of Koblenz-Landau. 1991.