Test Suites for Quality Evaluation of NLP Products

Frederik Fouvry and **Lorna Balkan**
Department of Language and Linguistics, University of Essex
Wivenhoe Park, Colchester
Essex CO4 3SQ, UK
(fouvry|balka)@essex.ac.uk

Abstract

Test suites are a useful evaluation tool for developers and users of NLP products. The paper gives an overview of the TSNLP design and methodology and describes how the TSNLP data and methodology can be used in practice to provide a reliable assessment method of the linguistic capabilities of NLP products.

1 Introduction

Test suites have long been recognized as a useful evaluation tool for NLP products, especially for the developer. Until recently, however, there have been few publicly available test suites or guidelines for their construction and use. The EU-funded project TSNLP (Test Suites for Natural Language Processing, LRE 62-089),[1] has filled that gap by providing a methodology for the construction of test suites, a substantial multilingual test suite (over 4,500 test items in each of its three languages, French, German and English), mounted on a database, and a number of tools for use and construction. The test data are general purpose, i.e. not specific to a particular domain or text type, and they focus on syntactic phenomena. They have been validated in a testing phase where a parser, a grammar checker and a controlled language checker have been tested. This paper describes the data, tools and methodology, and shows how the test suite can be used in practice to provide a reliable assessment method of the linguistic capabilities of a system.

2 Design of Test Suites

By *test suite* is meant a more or less systematic collection of specially constructed linguistic expressions (*test items*, e.g. sentences), perhaps with associated annotations and descriptions. Most existing publicly available test suites have been written for specific systems or simply contain numbers of "interesting examples". They have only limited or very specific annotations. For test suites to be reusable—since there are few publicly available test suites around, and hence a lot of work is duplicated by writing suites for specific system—there must be annotations to provide access via many different search keys.

This requirement leads us towards a large, systematically constructed test suite that can be applied to a wide variety of applications. The HP test suite (Flickinger *et al.* 87) went some way towards that goal. However, its usability and extendibility is restricted by the lack of supporting documentation and annotations. The DiTo test suite (Nerbonne *et al.* 91) pioneered more extensive annotations on test items and developed a database for the storage and retrieval of test items. TSNLP builds on the work of DiTo and is unique, to our knowledge, not only in terms of its size, but also in the richness of its annotations.

In TSNLP we tried to optimize features that test suites possess, that are unobtainable in test corpora, the other common, complementary testing technique. These features include amongst others things systematicity, non-redundancy, control over test data and explicit annotations. TSNLP has however, some specific characteristics which also influenced its design, which we look at first.

[1] We hereby acknowledge our partners: Dominique Estival, Kirsten Falkedal, Sabine Lehmann, Hervé Compagnion (ISSCO, Genève, CH); Klaus Netter, Stephan Oepen, Judith Klein, Judith Baur (DFKI, Saarbrücken, D); Sylvie Regnier-Prost, Eva Dauphin, Véronika Lux (Aerospatiale, Paris, F); Siety Meijer (formerly of University of Essex, Colchester, UK).

2.1 TSNLP-specific Characteristics

TSNLP aims to be:

Reusable The TSNLP test suite is designed to be a reusable resource for the obvious reasons given above. It achieves this in a number of ways. Since it was noticed that the reusability of many existing publicly available test suites was hampered by their lack of annotation, the data are extensively annotated and documented, with the provisions to allow the user to add his or her own annotations and extensions. Also in order to optimize reusability, the data are stored in a database.

Multi-purpose This means that the data are intended to be relevant to a wide range of applications and different evaluation scenarios. As we mentioned above, parsers, grammar checkers and controlled language checkers were a particular focus of the project, and some customised data was written for these applications. The different evaluation scenarios that the data is intended to support include:

- *diagnostic evaluation*, which aims at localizing deficiencies in the system;
- *progress evaluation*, for a comparison between successive stages of development of a system over a period of time;
- *adequacy evaluation*, which determines whether and to what extent a particular system meets some pre-specified requirements.

Test suites are most suitable for diagnostic evaluation, but diagnostic evaluation may form part of the other types of evaluation. For example, diagnostic evaluation may form part of an adequacy evaluation if one of the aims is to gauge the improvability of a system.

Multi-user Diagnostic evaluation is typically associated with developers. But users also need to perform some type of diagnostic evaluation. They may be, but are not necessarily, linguistically naive. With this in mind, care was taken to make the linguistic terminology of the test suite as accessible as possible to a wide audience through extensive documentation.

Multi-lingual The test data are available in three languages: English, French and German. They are parallel in the sense that they cover the same range of phenomena, but this parallelism does not extend to depth nor breadth of coverage. This is due to the relative importance of some phenomena in one language compared to another. Parallelism is also achieved in the naming system for the high level classification scheme and for the category and function names (with language-specific additions, where appropriate).

2.2 General Design Issues

2.2.1 Control

The main characteristic of a test suite is control over the test data. This is contrasted with test corpora where input comes from real life, i.e. data is linguistically complex and frequently messy, and it is difficult to isolate just the constructions one is interested in testing. Test suites allow one to focus on some phenomenon or combination of phenomena, by reducing the amount of extraneous material. This was achieved in TSNLP by following measures:

1. Restrictions:
   - Avoid ambiguity in the test items.
   - Keep the parameters that are not being tested, constant (e.g. unmarked word order, do not use unnecessary adjuncts, only use declarative sentences, ...). This is to avoid possible misinterpretations by the users.
2. Explicitness: Another way of controlling test data is to record the other phenomena present in the test item that are not explicitly being tested for. When one makes the data writers focus on different parameters, inadvertent complications are less likely to occur. And again, in this way it is easier to keep parameters constant and items simple. For instance, there is the *presupposition*-field in the annotations, which, as its name suggests, contains the phenomena presupposed to be correct for that test item, which are not explicitly being tested for. It also gives an indication of how complex an item is, and is as such a guideline for selecting the items.
3. Progressivity: is the principle of starting from a test consisting of simple items and using/writing increasingly more complex ones. This can be derived among other things from the already mentioned *presupposition*-field.
4. Systematicity: Test suites allow the possibility of systematic examination of every variation of a phenomenon or of a combination

of phenomena. In real life corpora one is unlikely to encounter the full range of variations that make up a phenomenon.

Systematicity is closely linked to the notion of exhaustivity. A test suite which exhaustively covers all the sub-phenomena of a phenomenon, or that exhaustively treats all the possible variations of the parameters which make up a phenomenon, can be described as systematic. One way in which TSNLP achieved exhaustivity was to cover exhaustively certain closed class items of a language, like pronouns, question words and auxiliary verbs.

Systematicity was greatly enhanced by the use of the annotation scheme and the data construction tools, which we discuss in greater detail below.

2.2.2 Ill-formed items

Ill-formed items in the test suite are derived from well-formed items. Negative examples are a particular focus of TSNLP since they are of direct relevance to applications such as grammar checkers, which are designed to identify and respond to ill-formed input. They are also necessary to check the robustness and possible overgeneration of other systems, such as parsers. Problematic however is the fact that there are many ways to derive an ill-formed test item from a well-formed one, which makes it extremely difficult to be both systematic and exhaustive. In TSNLP ill-formed items are derived from well-formed items in a systematic way by controlled use of the four operations standardly used to produce variations of sequences, namely:

- Replacement (French)
  L'ingénieur vient.
  ** L'ingénieur viens.*

- Deletion (German)
  Der Manager hält den Vortrag.
  ** Der Manager hält.*

- Permutation (French)
  Le travail.
  ** Travail le.*

- Addition (English)
  She can come.
  ** She can it come.*

Care should be taken that the test item created by one of these operations is not itself well-formed (e.g. if the verb can have fewer arguments, with the same or a different meaning). Because these cases are sometimes unavoidable, we used for these items a special grammaticality value, viz. *relatively ungrammatical*. This means that it is ungrammatical in this particular context of the test suite.

So TSNLP errors are limited to *competence* errors, that is, errors that violate normal grammatical feature restrictions (e.g. the wrong case marking on a noun, or the wrong number on a verb). Another error type, which we call *performance* errors, occur as a result of text manipulation, treatment of formatted structures, punctuation, etc. These are not treated in TSNLP since error corpora that contain such errors were not available to us. Performance errors are however of relevance to applications such as grammar checkers and should be investigated as part of the customisation process (see section 4.1 below).

2.3 The Annotation Scheme

Essential to the design of TSNLP was the construction of an efficient annotation scheme. The reusability of existing test suites is frequently hampered by their lack of adequate annotations. A key feature of the TSNLP annotation scheme is its flexibility. While the bulk of the TSNLP data are general purpose, the annotation scheme is sufficiently flexible to accommodate application and user specific annotations.

Annotations in TSNLP fall into four different classes: *item*, *phenomenon*, *set*, *user&application profile*.

The *item* annotations contain categorial and structural information that is independent of the phenomenon and user parameters. It contains information about the actual string of words, bookkeeping information, length, category, functional analysis and well-formedness. A common naming system for category and function names as well as high level phenomena names is adopted to ease cross-reference across languages.

The *phenomenon* annotations provide a hierarchical classification of the phenomenon (*phenomenon name* and *supertypes*) as well as information about the "presuppositions" of the phenomenon, and information about the source and type of the error for ill-formed items.

The *set* annotations allow one to group test items in any combination one wishes (e.g. one can group the positive and ill-formed variants of a particular test item).

The *user and application profile* annotations record the centrality of a phenomenon in general language, as well as its relevance for a particular application type and its frequency and weighting for a particular system and text type. The determination of these last values fell outside the scope of TSNLP but is discussed briefly in section 4.3. It is also recognized that the user might need to add other, system specific annotations of his own to record information such as system output, rule violations, etc. Facilities are provided for the addition of user-specific attributes in the ANSI C and Access versions of the database (see below).

The annotations are helpful to the test suite user in a number of ways. Firstly the many dimensions along which the test item is described (well-formedness, length, constituent structure, functional structure, phenomenon type, etc.), allow the user to extract very specific subsets of the data. One can, for example, extract all well-formed instances of test items that illustrate NP_Complementation and that contain a PP. Secondly, as mentioned above, the *presupposition* attribute serves to guide the user in the order in which items should be applied. Thirdly, groupings of items may give hints to the user about what is going wrong with the system by coinciding of groupings of errors. The ill-formed items are annotated with the instance, position, and optionally, comment on the type of error. The *set* window also allows the test suite writer to associate ill-formed test items with the well-formed counterpart from which it has been derived. This can be a guide to what the correct output should be (which can be useful for checking grammar checker output, for example), but it should be remembered that an ill-formed item can potentially be derived from a number of well-formed items, not all of which may be listed in the set value.

The annotations are not all of equal value: some are more valuable to test suite writers, others more to users, or to specific systems. But they all have the same purpose: to make the test suite reusable by providing a context for interpretation of the data.

3 A Methodology for Test Suite Construction

The design phase of TSNLP resulted in a methodology for test suite construction that was followed during the data creation process. This methodology was subsequently revised in the light of the testing phase. A full description of the methodology can be found in the TSNLP User Manual Volume 1 (TSNLP96a).

Phenomenon	English		French		German	
Sentence Types	75	100	389	387	107	14
Diathesis/Passive	154	126	176	119	147	148
Tense Aspect Modality	157	39	77	275	188	134
Complementation	158	890	209	618	218	246
Modification	300	482	329	63	131	67
Agreement	269	1050	376	1265	523	1907
Word Order			7	7	103	226
Coordination	147	186	379	319	105	429
Negation	289	129	68	100	110	145
Extra-grammatical	25	36				
TOTAL	1574	3038	2010	3153	1632	3316

Figure 1: Status of the data construction end of December 1995: relevance of individual phenomena differ for the three languages. The first column contains the number of grammatical items, the second the number of ungrammatical ones (including dubious and relatively ill-formed items).

The final methodology is couched in terms of the TSNLP annotation scheme. Here we discuss the relevance of annotations to the test suite builder.

The first step in creating a systematic test suite is to classify the item according to the phenomenon or subphenomenon it belongs to. Here, the hierarchical classification scheme was used. The list of phenomena for TSNLP (together with some distribution figures) is given in fig. 1. Each phenomenon is subdivided according to its domain, i.e. the syntactic category of the phrase in which it occurs, e.g. the adjective–noun construction is classified as NP_Modification.

How the phenomenon is subdivided after that is up to the test suite writer.

The attributes associated with ill-formed items allow one to record the position and instance of the error. For example, the error in *She can it come.* is identified as being at position 2:3, and the error instance is the word *it*. There are also so-called attribute-value slots in the error fields provided if one wants to formalize this error type (e.g. attribute:*number of complements*, value:*1*), though this proved rather difficult to define in practice. Additionally, or alternatively, a comment field is provided for the test suite writer to provide his own gloss of the error.

The data and actual annotation practices are documented in (TSNLP96d).

3.1 Use of Tools

The construction of test data is both difficult and time consuming. A number of prototype tools have been developed within TSNLP to aid data construction. At present these three construction tools are stand alone. A fuller description of the test suite generation tool and the lexical replacement tool can be found in (Arnold *et al.* 94) and (TSNLP96c), and tsct and tsdb (all versions) are described in (TSNLP96b).

3.1.1 tsct

The Test Suite Construction Tool (tsct) was developed because the data was created prior to the database. It writes the data in a machine readable format which is then normalized for the database (with an import script). It divides the data into four sets of annotations: *item*, *phenomenon*, *set*, *user&application profile*. Each set is visible to the user in the form of a window that serves as an input mask. The main advantage of the tool was that it freed the data writers of the burden of ensuring machine readability. The values of some attributes are calculated automatically, such as the length of a string or the instance of the substring occupying a position in the string. It also serves as a browser for already created data. A future possible enhancement would be to build in more intelligent functions, so that it might suggest, for example, an analysis of the string on the basis of existing test items.

3.1.2 Auto-TSG and Lexical Replacement Tool

The idea of automatically generating test suites was first proposed by (Arnold *et al.* 93), motivated by the fact that, despite the difficulties in test suite construction, some of which have been discussed above, much of the construction process is repetitive and boring. The repetitive nature of the task, and the need for systematicity suggested that at least some of the processes involved in test suite construction could (and should) be automated.

Thus a prototype grammar-based *Test Suite Generator* (Auto-TSG) was developed within the project, before the implementation of the database, and tested during the data writing. The user needs to write a DCG-like grammar, which will generate the items. We concluded that the Auto-TSG certainly can improve the systematicity of the test suite, and it proved very useful for full coverage of closed class items. A better integration with tsct would have been desirable.

TSNLP concentrates on the production of core phenomena, not specific to any particular domain or text type. In order to use it in a specific evaluation set up, the user may be required to customise the data to his application, text type and or domain. Part of this customisation process includes

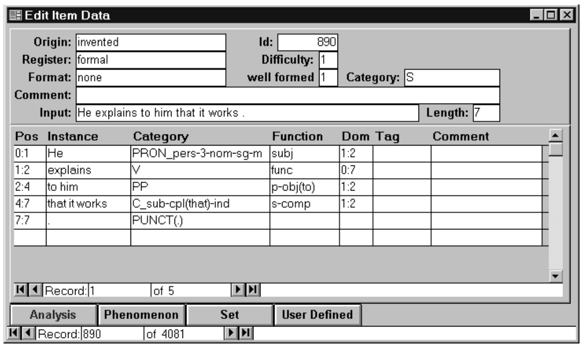

Figure 2: The Main Window of tsdb_3 (MS Access)

lexical replacement. TSNLP explored how this process may be automated, by implementing a small emacs-based *Lexical Replacement Tool*.

The usefulness of the tool depends to a large extent on the size and quality of the dictionary which it consults.

Other possibilities for lexical (and phrasal) replacement include aligning test items with corpora on the basis of tags (see section 4.3).

3.1.3 The Database

The database tsdb is implemented in three versions, ANSI C (tsdb_2), FoxPro (tsdb_1) and MS Access (tsdb_3). The latter two are built in commercially available systems, and have the advantage of user friendly graphical interfaces (see e.g. fig.2).

The ANSI C version allows retrieval examples to be formulated a simplified SQL-like query language.[2] For example, the following query finds all grammatical strings that illustrate NP_Agreement:

```
retrieve i-input where i-wf = 1 & \
        p-name = "NP_Agreement".
```

All three versions use the same format to store the data. tsdb_3 uses a QBE-querying style for most of the queries, but the user still gets the possibility to design his own specific queries.

4 Applying the Test Suite

To validate the TSNLP data, tools and methodology a number of evaluations on actual systems were carried out within the project, using different types of applications (a parser, grammar checker and controlled language checker). The evaluation set ups ranged from black box (in the case of the grammar checker) to glass box (in the case of the parser).

[2]It is also accessible at `http://tsnlp.dfki.uni-sb.de/tsnlp/tsdb/tsdb.cgi`

The idea behind the testing phase of TSNLP was to see how useful the set of chosen annotations was. A particular focus of the testing phase was also to see how easy it is to customise the general data to specific systems or corpora.

4.1 Customisation of Data

A general test suite needs to be tailored for a specific system. This process is called customisation. The customisation process includes adapting the suites to lexical requirements, formatting requirements, adding new phenomena and items, adding output attributes to the test suite. This is made easier by the annotations: the required phenomena are easily retrieved, and the function of every sentence is clear from the set and phenomenon it belongs to. Once a suitable replacement is found for a word that does not belong to the lexicon, the test suite is adapted relatively easily.

When new items had to be created to extend the grammatical coverage of the suite, the writers were guided by the adaptations that were already done.

The advantage of having the possibility of adding system specific fields was clearly proven by the German site, who developed an interface between the application they were testing and tsdb, thus allowing automatic comparison of system output and test suite data.

4.2 Using the TSNLP Test Suites

The results were very encouraging, showing how the systematicity and rich annotations of the test items facilitated the detection of system errors. The fact that the data are mounted onto a database means that they can be more easily manipulated and retrieved.

The classifications, annotations and systematicity of the test suites enabled hypotheses to be made about errors in the systems, and to define criteria to make new tests for more refined results. For the testing of the grammar checker for instance, the tests threw light on consistency of the system, lexical deficiencies, morpho-syntactic deficiencies and they also revealed that the checker did much worse for ill-formed examples than for well-formed ones. In the end, detailed conclusions could be drawn from the tests, and they provided a good first impression of the performance of the systems.

4.3 Tuning Test Suites to Corpora

In order for a test suite to be maximally useful to the end-user, one needs to be able to relate phenomena in the test suite to phenomena occurring in one's own corpora. That is, one needs to be able to establish the frequency of item types in a corpus.

A *tag* slot is provided in the annotation scheme which potentially can be exploited for aligning test suites to corpora. Customisation of this type was not however carried out, although some work on aligning test suites to corpora was done within the project (Balkan & Fouvry 95; Dauphin *et al.* 95), mainly on a manual basis. While the automation of such a task is clearly desirable, we observed that most existing corpus analysis tools concentrate on a morphological parsing of large amounts of texts, identification of existing or potential domain specific terminologies, and statistical records on the lexical content of corpora.

5 Conclusion

TSNLP developed guidelines, methodology and tools for test suite use and construction, and a substantial annotated general purpose test suite in three languages. These results are immediately usable and useful for any one working with NLP applications and evaluating them. TSNLP is thus set to become a standard model for test suite design and construction, and the data and tools will hopefully become widely used in the NLP community to help developers and users alike in quality assessment of NLP products.

The TSNLP results, including data, software and reports and user manual, are available from `http://tsnlp.dfki.uni-sb.de/tsnlp/ftp.html` from the ftp site: `anonymous@tsnlp.dfki.uni-sb.de/tsnlp`, or from the authors.

References

(Arnold *et al.* 93) Doug Arnold, D. Moffat, Louisa Sadler, and Andy Way. Automatic test suite generation. *Machine Translation*, 8:29–38, 1993.

(Arnold *et al.* 94) Doug Arnold, Martin Rondell, and Frederik Fouvry. Design and implementation of test suite tools. Report to LRE 62-089 D-WP5.1, University of Essex, UK, December 1994.

(Balkan & Fouvry 95) Lorna Balkan and Frederik Fouvry. Corpus-based test suite generation. Report to LRE 62-089 D-WP5.2, University of Essex, UK, October 1995.

(Dauphin *et al.* 95) Eva Dauphin, Véronika Lux, Doug Arnold, Lorna Balkan, Frederik Fouvry, Sylvie Regnier-Prost, Judith Klein, Klaus Netter, Stephan Oepen, Dominique Estival, Kirsten Falkedal, and Sabine Lehmann. Checking against corpora. Report to LRE 62-089 D-WP3.2, University of Essex, UK, April 1995.

(Flickinger *et al.* 87) Dan Flickinger, John Nerbonne, Ivan A. Sag, and T. Wasow. Toward evaluation of NLP systems. Distributed at the 24th Annual Meeting of the Association for Computational Linguistics (ACL), Stanford, 1987.

(Neal & Walter 91) J. Neal and S. Walter. Natural language processing systems evaluation workshop, Berkeley. Technical Report RL-TR-91-362, Rome Laboratory, New York, 1991.

(Nerbonne *et al.* 91) John Nerbonne, Klaus Netter, A. Kader Diagne, Judith Klein, and L. Dickman. A diagnostic tool for German syntax. Technical Report D-92-03, DFKI, Saarbrücken, 1991. Also in (Neal & Walter 91).

(TSNLP96a) TSNLP. University of Essex, ISSCO, Aerospatiale, DFKI. *User Manual. Volume 1: Background, Methodology and Guidelines*, April 1996.

(TSNLP96b) TSNLP. University of Essex, ISSCO, Aerospatiale, DFKI. *User Manual. Volume 2: Core Test Suite Technology*, April 1996.

(TSNLP96c) TSNLP. University of Essex, ISSCO, Aerospatiale, DFKI. *User Manual. Volume 2b: Test Suite Technology*, April 1996.

(TSNLP96d) TSNLP. University of Essex, ISSCO, Aerospatiale, DFKI. *User Manual. Volume 3: Test Data Documentation*, April 1996.