TSNLP

Test Suites for Natural Language Processing

User Manual

Volume 1

Background, methodology,
customisation and testing

University of Essex, UK
ISSCO, Université de Genève, Switzerland
Aerospatiale, France
DFKI, Germany

Preface

The TSNLP user manual summarises the results of the EU-funded project TSNLP (LRE-62-089, Test Suites for Natural Language Processing), which started in December 1993 and had a duration of 29 months. The project produced a methodology and tools for test suite construction in addition to substantial test suites in three languages - English, French and German (over 4000 items in each language). The test material covers central syntactic phenomena in the three languages and aims to be relevant to any developer or user of systems with grammar components who wish to test, benchmark or evaluate those systems. To allow for flexible access to the test suites they are mounted on a database.

The project consortium consisted of the following centres and principal researchers:

- University of Essex, UK (Coordinators):[1]
  Doug Arnold, Lorna Balkan, Frederik Fouvry
- ISSCO, Switzerland:[2]
  Dominique Estival, Kirsten Falkedal, Sabine Lehmann, Hervé Compagnion
- Aerospatiale, France:[3]
  Eva Dauphin, Veronika Lux, Sylvie Regnier-Prost
- DFKI, Germany:[4]
  Klaus Netter, Stephan Oepen, Hans Uszkoreit

Other contributors included Evangelia Kordoni, Siety Meijer and Martin Rondell (University of Essex), and Judith Baur, Tom Fettig, Judith Klein, Fred Oberhauser (DFKI).

The user manual is in three volumes:

- Volume 1:
  Background, methodology, customisation and testing
  Editors: Lorna Balkan, Frederik Fouvry, Sylvie Regnier-Prost

[1]University of Essex, Wivenhoe Park, Colchester C04 3SQ, UK
[2]ISSCO, Université de Genève, 54, route des Acacias, CH 1227 Genève
[3]Aerospatiale, Centre Commun de recherches Louis-Blériot, Département Information Documentation, 12, rue Pasteur BP 76, 92152 Suresnes, Cédex, France
[4]DFKI, Stuhlsatzenhausweg 3, D 66123 Saarbrücken, Germany

- Volume 2
  Test suite technology
  Stephan Oepen, Frederik Fouvry, Klaus Netter, Tom Fettig, Fred Oberhauser, John Gregory.

- Volume 2b
  Test Suite Technology
  Editor: Frederik Fouvry

- Volume 3
  Test Data documentation
  Sabine Lehmann, Dominique Estival, Kirsten Falkedal, Hervé Compagnion, Lorna Balkan, Frederik Fouvry, Judith Baur, Judith Klein.

Volume 1 contains a Background chapter in which some of the factors which have influenced the design of the project are sketched.

The Methodology chapter gives a step by step account of how one can go about writing core data, that is, data that cover central phenomena of a language and that are intended to be applicable to a wide range of applications.

The Customisation chapter describes how the core data can be customised to a particular application (and sketches how it could be customised to a particular domain or text type).

The chapter on Testing gives an example of how the test suite can be applied to a real life evaluation scenario.

Volume 2 contains a description of the annotation scheme on which the data was constructed, the construction tool tsct used to create the data, the database tsdb on which the data is mounted.

Volume 2b contains a description of and documentation on the automatic test suite generation tool (Auto-TSG) and the lexical replacement tool.

Volume 3 contains the detailed documentation that accompanies the data, and which is intended to make the data more accessible to users. It also contains the category and function labels used in the English, French and German test data with examples for each language, in addition to the vocabulary list used for the test data by the three languages.

TSNLP reports and this User Manual are available from the Coordinator at the following address:

L. Balkan,
CL/MT Group,
Department of Language and Linguistics,
University of Essex,
Wivenhoe Park,
Colchester,
CO4 3SQ
UK

Fax
+44 1206 872085

email
balka@essex.ac.uk

TSNLP deliverables, data and software can be accessed at the following address:

http://tsnlp.dfki.uni-sb.de/tsnlp/ftp.html

or via the ftp site:

anonymous@tsnlp.dfki.uni-sb.de/tsnlp

for those who have ftp but not world-web access.

Overview

This volume consists of four chapters. The Background chapter discusses the factors which influenced the design of the project.

The Methodology chapter gives a step by step account of how one can go about writing core data, that is, data that cover central phenomena of a language and that are intended to be applicable to a wide range of applications.

The Customisation chapter describes how the core data can be customised to a particular application (and sketches how it could be customised to a particular domain or text type).

The chapter on Testing gives an example of how the test suite can be applied to a real life evaluation scenario.

Contents

Chapter 1

Background

1.1 Introduction

In TSNLP we have tried to optimize certain design features that we identified as desirable through the study of existing publicly available test suites (see [10]) and literature relating to test suites (see [4]). These are reflected in the annotation scheme we adopted (Volume 2 of the user manual), in the design of the database (see Volume 2) and in our methodology (see the next chapter of this volume). TSNLP has however, some specific characteristics which have also influenced its design, and we begin by reviewing these.

1.2 TSNLP-specific characteristics

The TSNLP has the following characteristics:

- Multi-purpose
  By "multi-purpose", we mean that the data are intended to support different types of testing and evaluations, performed for a variety of different purposes.[1] As suggested by (inter alia) the EAGLES subgroup on evaluation [9], the following purposes can be distinguished:

  - *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 typically associated with developers, who perform diagnostic evaluation, usually with glass-box access to applications or components. However, they are also relevant to other types of evaluation.

[1] A number of issues arising from this diversity are discussed in the background document "Issues in Test Suite Design" [4].

Adequacy evaluation attempts to find out whether a system is suitable for a specific application. This may amongst other things involve the identification of the deficiencies of a system, which requires diagnostic evaluation. And since many NLP products are customised on user demands, progress evaluation could also be an essential part of adequacy evaluation.

By "multi-purpose" we also mean that the test suite should be relevant to a number of applications. The bulk of the TSNLP test data is intended to be relevant to any system that has an important grammar component. Some application-specific data was also written (for parsers, grammar checkers and controlled language checkers).

The annotation scheme is designed to be sufficiently flexible to accommodate not only application specific but also user specific test data with appropriate annotations. This is also a reflection of the assumption or postulate which underlies the TSNLP approach to test suite design, namely that any test suite to be used for some given test or evaluation has to be more or less specific in order to yield optimally informative and interpretable results, given that at any stage of development a NLP system has features which make it unique and specific, and all potential users of such NLP systems have specific needs and requirements of various kinds.

- Multi-user
  The test suite is designed to be useful for developer and user alike, and many aspects of its design are user-oriented (such as the stress on detailed documentation of the data).

- Multi-lingual
  The test data are available in three languages: English, French and German, and have been produced in parallel, in the sense that they cover the same range of phenomena, and use the same high level classification and the same category and function names (described in the section on annotations in Volume 2). It is of course possible to extend the test suite to other languages.

- Reusable
  The TSNLP test suite is designed to be a reusable resource. In order to enhance reusability, the traditional notion of a test *suite* as a monolithic list of test items has been abandoned in favour of the notion of a relational database, where test items created for different purposes are stored and associated with a detailed set of annotations and from which purpose specific and application specific sets of test items can be extracted by searching on specific annotation types. As a certain amount of customisation must frequently be expected (as discussed in more detail in the chapter on customisation below), reusability is further enhanced by the provision of tools to replace lexical material with user or application specific equivalents, as well as of tools to generate additional test items (see Volume 2b and the next chapter of this volume).

1.3 General Design Issues

In [4] we reviewed some of the advantages of test suites over test corpora (systematicity, non-redundancy, control over test data, explicit annotations, etc). The way in which TSNLP aims to optimize these various qualities of test suites is discussed below:

1.3.1 Control over test data

In contrast to test corpora which contain real life data, central to the notion of test suite is the use of *artificially constructed* test items, built so as to test for a specific phenomenon or combination of phenomena in a controlled fashion.

What makes test suites valuable in comparison to corpora is that they can focus on specific linguistic phenomena and present them in isolation. Of course, one cannot say that any sentence illustrates just some phenomenon, since language is inherently a combination of different words standing in specific grammatical relations. Thus, talk about presenting a single phenomenon can only be understood in the sense that as many linguistic parameters as possible are being kept under control. This holds in particular for those cases where a phenomenon is illustrated by systematic variation over the parameters making up this phenomenon, while all other parts of the test items are being kept constant.

Vocabulary is one of these parameters that need to be controlled. Where the phenomenon that is being tested is not lexical in nature, it is advisable to keep the vocabulary as limited and stable as possible. Categorially and semantically ambiguous words should be avoided as far as possible and only included and used where ambiguity is tested for explicitly. This is the strategy adopted by TSNLP. The vocabulary of TSNLP has been mainly taken from the business domain and is kept as small in range as possible.

TSNLP test items are also kept deliberately simple (avoiding for example modifiers and adjuncts) in order to reduce the interaction problem.

As noted above, however, it is usually impossible to test any phenomenon entirely in isolation. The TSNLP annotation scheme, however, allows one to keep track of other phenomena present in the test item that are not explicitly being tested for (through the *presupposition* attribute, which, as its name suggests, contains a (list of) phenomenon(a), which are considered "presupposed" when using a particular test item).

1.3.2 Progressivity

A useful property of test suites is the notion of progressivity. Progressivity is the principle of starting from simple test items and increasing in complexity. Progressivity can be achieved by ensuring that each test item contains only one phenomenon (or rather, subphenomenon or even feature) which distinguishes it from any other test item. This principle should not only reinforce systematicity during test data construction but also enable future test data users to apply the test data in a progressive order. This progressive order of application of the data should in turn render precise identification of system coverage and deficiencies easier: if a system fails on a test item but has performed successfully on previous items, the failure must be due to the new subphenomenon or feature introduced in that new test item.

Progressivity is achieved in TSNLP also by the use of the *presupposition* attribute. Progressivity is assured if:

- The phenomena listed in this field are covered by separate (sets of) test items. These may use identical input strings (eg. the same string can be used to illustrate C_Complementation and C_Agreement), or more likely substrings, but the annotations for the particular subphenomena they represent may be different (eg an NP used in NP_Agreement will be analysed into a determiner and noun, but treated simply as an NP inside C_Complementation).

- The (sets of) test items for phenomena which are listed as *presuppositions* are submitted to the NLP systems under test before the test items for which they are presupposed.

1.3.3 Systematicity

Systematicity in test suites can take several forms. It is closely linked to the notion of progressivity. It is also 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. We describe below some ways in which we achieved systematicity and some degree of exhaustivity in TSNLP. Systematicity was greatly enhanced by the use of the annotation scheme and the data construction tools, in particular the test suite generation tool (see Volume 2b and the following chapter of this volume).

Closed Class Items

As was suggested during the mid-term Peer Review, one way to ensure that the database created in TSNLP will meet the demand for reusable test material is to cover exhaustively at least some of the type of input that most users are likely to want to check. In particular, the TSNLP vocabulary should cover the closed class items of a language, and test items should be constructed to test exhaustively for their proper treatment.

The exact definition of a “closed class” is, however, open to interpretation. The list TSNLP agreed on is given in the following chapter of this volume.

Ill-formed items

Negative examples are a particular focus of TSNLP for the following reasons. Not only can they be interesting from the developer's point of view, since the indiscriminate acceptance of an ungrammatical construction often bears the risk of misinterpreting a grammatical construction, but they can be of direct relevance to applications such as grammar checkers, which are designed to recognize and mark such constructions.

It is useful to distinguish between two types of ill-formed input:

- Test items that violate normal grammatical feature restrictions (e.g. the wrong case marking on a noun, or the wrong number on a verb). We shall refer to this type of error as *competence* errors.
- Test items that contain errors that occur as a result of text manipulation, treatment of formatted structures, punctuation, etc. We shall refer to this type of error as *performance* errors.

Error-types in the TSNLP data are restricted at present to competence errors, since the coverage of performance errors presupposes the availability of some type of error corpora, which were not available to us. [2]

Ill-formed items are derived from well-formed items. However, 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 4 operations standardly used to produce ill-formed data, namely changes inside elements (e.g. change of number of a subject NP), addition of elements (e.g. addition of an NP in a sentence with an intransitive verb), deletion of elements (e.g. deletion of an obligatory object) and permutation of elements (e.g. by inverting the order of the verb and the object). Examples of these operations are given in the following chapter of this volume.

[2] Performance errors are however of relevance to applications such as grammar checkers and should be investigated as part of customisation.

Chapter 2

Methodology

2.1 Introduction

This chapter describes a methodology for test suite construction that was followed for the production of core TSNLP data, *i.e.* test items that are applicable to a range of applications, including parsers, grammar checkers and controlled language checkers. It refers to attributes (indicated in italics) which can be found in the TSNLP annotation scheme described in Volume 2. Motivation for the test suite methodology is given in the Background section in this volume.

It is assumed that the user will be producing the data using the tsct test suite construction tool (described in Volume 2), and some familiarity with this tool is presumed, such as its division of annotations into 4 "windows": *item*, *phenomenon*, *set* and *user & application profile*. Certain parts of the methodology can be automated, and thus a section on the use of tools in test suite construction is appended to this chapter.

2.2 A procedure for test suite construction

Below, we suggest a procedure that can be followed (not necessarily in the order given) for the production of test suites following the TSNLP model.

1. Choose a phenomenon and subphenomena to treat
   Our choice of phenomenon and subphenomenon to treat in the core data was determined by their centrality to the language. Other criteria are possible, such as relevance to a particular application or the specific system under test, or frequency in a certain text type. Where interaction of phenomena is being tested for, the choice of which combinations of phenomena to treat should be made following similar criteria.

2. Describe the phenomenon and subphenomena
   In order to treat the phenomenon systematically, it is important to describe it in adequate detail. Different sources can be used, including reference grammars such as Quirk *et al.*, *A Comprehensive Grammar of*

the English Language; [12] (for English);Duden, *Grammatik der deutschen Gegenwartssprache* [8] (for German) and Grevisse, *Le bon usage* [11] (for French), or implemented grammars of a system. Additionally, existing test suites can serve as an extra checklist. For a syntactic test suite such as TSNLP, it is useful to identify the semantic/pragmatic factors that interact with the phenomenon.

The description of the phenomenon and subphenomena provides initial input for the documentation of the test items, which will subsequently have to be revised (see step 11). This step also provides an initial classification of the phenomenon and subphenomena which can serve as values for the *name* and *supertype* attributes of the test item (see section 2.3 below) and provide possible values for *parameters*.

3. Determine the form of test items
   The form or category of a test item is normally the domain in which the phenomenon or subphenomenon occurs (e.g.. NP for NP_Agreement, C for C_Negation), but it may be that the phenomenon is better illustrated by being contained in a higher domain. An example might be to illustrate non-restrictive relative clauses at clause rather than NP level, to better illustrate the role of punctuation within this phenomenon.

4. Decide on *names* and *supertypes* of test items

5. Decide on the closed classes, if any, that test items will cover
   See section 2.4 below. The choice of which closed classes to expand may be influenced by the set of closed classes expanded in a related phenomenon.

6. Construct wellformed test inputs (sentences or sentence fragments) using, if appropriate, the lexicon given in Volume 3 and the General Guidelines given in section 2.5 below.

7. Annotate each test item for the following (in the tsct item and phenomenon windows):

   - bookkeeping information
     Bookkeeping information includes *author* and *date*
   - *ID*
     See section 2.6 below
   - *register, format, difficulty, interaction, purpose, restrictions*
     The default value for these attributes for the core data are listed in table 2.1.
   - *category*
     This is the form of the test item that has been decided in step 3 above.
   - *WF* (Well-formedness) and *length*
     Grammaticality judgments should be made by native speakers. The values are:

Attribute	Value
register	formal
format	none
difficulty	1
interaction	none
purpose	test
restrictions	neutral

Table 2.1: List of default values

1=correct

0=not correct

2=dubious

3=relatively ill-formed (w.r.t. the phenomenon)
For example, the string *a little firm* is relatively ill-formed as an example of a determiner-noun construction, but perfectly grammatical as an determiner-adjective-noun construction. Relatively ill-formed items should be avoided where possible.

Length is calculated automatically using `tsct`.

- *analysis*
  Use the *category* and *function* names given in Volume 3. The *instance* is calculated automatically from the *position* information.
- *presuppositions*
  If the items are well constructed, the number of co-occurring phenomena in a test item (presuppositions) should be as small as possible unless interaction is specifically being tested for.
- *parameters*
  Parameters can serve several functions: to provide additional information on how positive and negative items were generated (in which case all attributes including *attribute* and *value* slots are usually filled in), and/or to give an indication of the nature and position of an error in negative items (in which case only *position, value* and *instance* slots may be filled in).
- *comment*
  Comment slots are provided in various parts of `tsct` so that the test suite writer can insert any additional information that does not fit neatly into any other slot.

8. create ill-formed items.[1]
   Derive ill-formed test items from well-formed items using one or more of the operations given in section 2.7

[1]Alternatively, ill-formed items can be constructed in tandem with well-formed items (see step 6 above)

9. fill in *set* annotations
   This involves deciding what configurations of negative and positive test items one wishes to have. One useful organizing principle is to create sets consisting of a positive item and the negative item(s) derived from it. Another (for diathesis, for example) is to associate related items across different phenomena (e.g. relate passive items with their active counterparts in C_Complementation).

10. fill in *user & application profile* annotations
    This task can be reserved for customisation.

11. update documentation
    The documentation should give a clear indication of the breadth of coverage of a phenomenon in the data (*i.e.* the subphenomena covered), motivation for that choice, a description of the phenomenon and subphenomena, the choice of ill-formed data, the closed classes expanded in the data, any presuppositions not specifically annotated for, plus any other information that will be of use to the test suite user.

12. do consistency checks using tools
    Useful checks that can be performed on the data include checking the vocabulary used in the data (for misprints), checking the category and function lists, and comparing categories and their associated strings. Tools for performing these checks are described in section 2.8.2 below.

13. import the data using the import scripts
    The import scripts are described in 2.8.1 below.

2.3 (Sub)phenomenon names

The list of phenomena for TSNLP includes:

Sentence and clause types
Diathesis
Tense, Aspect, Modality
Complementation
Modification
Agreement
Word Order
Coordination
Negation
Parentheticals
Abbreviations

Each phenomenon is subdivided according to its domain, *i.e.* the syntactic category of the phrase in which it occurs. The list of domains we use for the TSNLP data is as follows:

S (sentence)
C (clause)
NP
AP
ADVP
PP

The two categories S and C are not equivalent, although they may sometimes seem to label the same input string, because a sentence (S) may be composed of only one clause (C). Nevertheless, they are different:

• A clause is defined as the syntactic phrase containing a verb (finite or non-finite) and all of its arguments (including the subject) and modifiers. On the other hand, a sentence is the whole unit starting with a capital letter and ending with a strong punctuation mark in normal written text.

• A sentence may contain several clauses, but it may also be composed of only one clause.

In our test data, the input for most test items is a single sentence, ending with a period. Future data might include test items whose input is a paragraph composed of several sentences.

• The category VP is included to account for VP complements, e.g. infinitives or participials. Its inclusion in the list of TSNLP categories does not imply that the TSNLP analysis of a clause must involve a VP level or follow a particular linguistic theory: the test data writers are free to use it or not to use it, depending on the particular language and the analysis they wish to adopt.

From the point of view of phenomenon description, 'subcategorization' (or 'valence') includes the subject, and so is better treated at the clause level (*i.e.* C_Complementation). Subject-Verb agreement is also a phenomenon which is treated at the clause level (C_Agreement).

The phenomenon names resulting from this first subdivision are as follows:

– S_Coordination
– C_Complementation
– NP_Complementation
– AP_Complementation
– NP_Modification
– AP_Modification
– C_Agreement
– NP_Agreement
– C_Word-order
– etc.

How the phenomenon is subdivided after that is up to the test suite writer. One possibility is to follow the French example of using *elements*. Elements indicate which parts of the domain are affected by the phenomenon. For example, N, Det and AP are the elements of NP_Agreement. Elements can be indicated either by their syntactic categories. For example, each of the following sets specifies different types of NP_Agreement:

– NP_Agreement_DET_N (agreement in a simple NP)
– NP_Agreement_DET_N_Adj (agreement involving one adjective)
– NP_Agreement_DET_N_AP (agreement involving a whole AP)
– etc.

As the examples given above show, the elements may be at different levels of syntactic analysis (e.g. AP and ADJ). Alternatively elements can be indicated by their function or by a mixture of both:

– C_Agreement_subj(NP)_V
– C_Agreement_subj(NP_quant)_V

Other characteristics besides the category and/or function of the elements may be indicated. A typical example is the valence values for verbs in C_Complementation, where the possible arities (zero-valent, mono-valent, divalent etc.) are added in front of the list of actual elements. This yields phenomenon names such as the following:

– C_Complementation_zero-valent_subj(PRON_impers)_V
– C_Complementation_mono-valent_subj(PRON)_V
– C_Complementation_mono-valent_subj(NP)_V
– C_Complementation_divalent_subj(PRON_impers)_V_obj(NP)
– etc.

2.4 Closed classes

The exact definition of a "closed class" is open to interpretation. It may be based on the distinction between "grammatical words" and "semantically full lexical items" (e.g. between modal verbs and main verbs in English), or on the notion of productivity (speakers do not create new pronouns or new prepositions, but often create new nouns, verbs or adjectives). However, the distinction may become blurry in some cases: prepositions are not semantically empty, and neither are modal verbs.

The actual extension of the closed classes in a particular language is thus a matter of definition and the final decision of which items to include in the closed classes is left to the data writers.

By way of example, the following can be considered as closed classes:

Other categories may be considered depending on the language.

2.5 General guidelines

- general:

CLOSED CLASS	instances
Pronouns	personal, reflexive, possessive, clitic, relative, etc.
Question Words	interrogative pronouns, complementizers
Determiners	articles, possessives, demonstratives
	(a subset of) quantifiers, etc.
Auxiliary verbs	copular verbs and modal verbs, where applicable
Conjunctions	coordination & subordination
Adverbs	negation,
	and the most common temporal/locative deictics
Prepositions	(maybe a subset of a larger open class)

– no ambiguities
Both categorial and syntactic ambiguity should be avoided where possible
– no modifiers and adjuncts
– unmarked word order

- sentence level:
  – declarative sentences
- verb form:
  – 3rd person singular
  – present tense
  – indicative
  – active
  – unambiguous verbal morphology
  – no auxiliaries/modals
- NPs:
  – explicit case marking where appropriate or possible
  – fixed definite determiners (no demonstratives, possessives etc. unless necessary); alternatively (simple) pronouns where possible
- PP:
  – no complex prepositions

Of course, these guidelines do not apply when the test data must specifically check elements which are otherwise excluded.

2.6 ID-numbers

To ensure consistency across the database and to provide the possibility of multi-lingual cross-references, we chose common *supertypes* for phenomena, and for ease of data writers, we put information about domain and phenomenon names into the ID numbers.[2] The beginning of the ID number consists of first the domain and the phenomenon number listed in the table below:

[2] At that stage there was no database available yet in the project.

Domain IDs :	S =	1
	C =	2
	NP =	3
	AP =	4
	ADVP =	5
	PP =	6
Phenomenon IDs:		
	S_Types =	1
	(Question, Imperative, Conditional)	
	Diathesis (Passive) =	2
	Tense, Aspect and Modality =	3
	Complementation =	4
	Modification =	5
	Agreement =	6
	Word Order =	7
	Coordination =	8
	Negation =	9
	Punctuation =	12
	Parentheticals =	13
	Abbreviations =	14

The ID number can be up to eight digits in length.

2.7 Ill-formed items

Ill-formed items can be derived from well-formed items by the application of one of the following operations:

- Replacement

  The operation of replacement is illustrated with test data for the phenomenon of Agreement.

  (1) a. (Fr.) *L'ingénieur vient.*
      b. (Fr.) ** L' ingénieur viens.*

  (2) a. (Fr.) *La femme.*
      b. (Fr.) ** Le femme.*

  Deriving ill-formed items from C_Agreement items (example 1) involves modifying the value of person or number. Deriving ill-formed items from NP_Agreement (example 2) involves modifying the value of gender or number. Example 1b illustrates wrong person, example 2b wrong gender.

- Deletion

  The operation of deletion is illustrated with test data for the phenomenon of C_Complementation.

(3) a. (Ger.) *Der Manager hält den Vortrag.*
b. (Ger.) ** Der Manager hält.*

Care should be taken that the test item created by deletion of an element (here, a complement) is not itself well-formed (e.g. if the verb can have fewer arguments, with the same or a different meaning).

- Permutation

  We illustrate the operation of permutation with the phenomenon of Word Order in French. The following are examples of NP_Word-order:

  (4) a. (Fr.) *De quoi parle-t-il ?*
  b. (Fr.) ** De quoi il parle ?*[3]

  (5) a. (Fr.) *Le travail.*
  b. (Fr.) ** Travail le.*

- Addition

  As with deletion, the operation of addition is illustrated with test data for the phenomenon of C_Complementation.

  Come subcategorizes for an infinitival complement. Therefore, the second example is ill-formed.

  (6) a. (Eng.) *She can come.*
  b. (Eng.) ** She can it come.*

2.8 Use of Tools

2.8.1 Tools to aid the construction process

Using the Automatic Generation Tool (Auto_TSG)

The Auto_TSG generates test items using a DCG-like grammar. It is the responsibility of the user to provide this grammar. As some experience is needed to use grammars successfully, we give an example of how it was used in the project. The tool is described in detail in D-WP5.1 [2]. A manual is given in Volume 2b.

The English group used the Auto_TSG amongst other things to generate templates for subtypes of C_Complementation.

Phrasal verbs in C_Complementation

Complementation in verbs governs number, category and case of the complements. To create ill formed items, we introduce a variation using these parameters (where possible), and we allow deletion and addition of arguments.

[3] This question is acceptable in the spoken language.

For example: a positive item for phrasal verbs is *He calls her up.* We can allow within C_Complementation[4] variation in subject and object as follows:

- category (e.g. PP *On him* instead of PRON *He* or NP):
  **On him calls her up.*
- case (e.g. *Him* instead of *He*):
  **Him calls her up.*

For the particle *up* the following can be varied:

- category (e.g. *ing*-form *working* instead of an adverb):
  **He calls her working.*
- wrong particle (e.g. *between*):
  **He calls her between.*

For the added argument we take *it*:

- **He call her up it..*

As a general word order template we assume "`subj verb obj particle punct`". All this gives us the following Auto_TSG grammar:[5]

```
sentence(phrasal) ---> subj, verb(Particle), obj,
                       particle(Particle), punct.
subj ---> ['He']; []; ['Him']; ['On him'].
verb(Particle) ---> [Verb], {v(Verb, Particle)}.
obj ---> [her]; []; [she]; [on, her].
particle(Particle) ---> [Particle]; []; [Particle, it].
punct ---> ['.'].

v(call, up).  % for positive items v(call, between).  % for negative
items (wrong particle) v(call, working). % for negative items (wrong
 % category)
```

This generates $4*3*4*3*1$ sentences. In general when using the Auto_TSG, it is best to calculate the number of generated sentences before you start otherwise the number of combinations can quickly reach several thousands.

Auto_TSG limits the number of successive rule applications in depth by maximum and minimum depth settings. Maximum depth stops infinite recursion, minimum depth determines the minimum number of rule applications (in the example grammar, this has to be less than or equal to 2, since `sentence` calls other rules). These settings are reflected in the generated tree by the depth of respectively the deepest and the shallowest node.

[4]Without violating C_Agreement because we are not testing subject-verb agreement, *i.e.* we cannot vary number.

[5]Terminal symbols are given as atoms in Prolog lists `[...]`, `;` means "or", variables start with a capital and `{...}` DCG-like "procedural attachments".

In this case the number of generated sentences will depend to a great extent on the lexicon given by the predicate `v/2`. This cannot be limited using above mentioned settings, since it influences only rule applications in breadth.

It is also important to keep the grammars rather simple, and limited in coverage: if you end up writing and testing grammars that are as complex as the one you write the test suite for, then the tool is not of much use.

After transforming the Auto_TSG file into a tsct file,[6] the unwanted items can be removed.

The main strength of the Auto_TSG is in the generation of a very systematic and large group of test items, all generated *exactly* following the same rules. Another advantage is that the grammar can be reused later on, unaffected or with some changes (to improve it, to change the lexicon, or to expand it). The grammar can thus be viewed as a "virtual" test suite.

However, some problems remain, for example, the removal of the unwanted items, which look very similar to others. One can either seek to eliminate them after they have been generated, or exploit the filter facility within the grammar (`filter/2`). A way of limiting this problem is by keeping the grammar small in the first place.

Note that although the sets generated by the Auto_TSG are systematic, it is time consuming to annotate them. It is better to generate at least part of the annotations along with the items. Using the predicates `add_note/1` and `remove_note/1` the user can add certain annotations to, or remove them from, the items to be generated.[7] In the absence of a more elaborate extension of that kind to the Auto_TSG, we developed a number of scripts, which are described in the section below.

Scripts for manipulating data

As tsct files have the format of a database (*i.e.* values separated by field separators, and records separated by record separators), `awk(1)`-scripts can be used not only to read, but also to write data.

- `tsct-format` takes two arguments: a lexicon file and a sentences file. The sentences file looks as follows:

  ```
  */Him/comes/.
  /He/peers/into it/.
  /the manager
  ?/He/has/been being seen/.
  ```

  The constituents are separated by a slash `/`. The place before the first slash contains an indication of the well-formedness of the item: `*` means ill formed, `?` dubious, and the empty string is for well formed sentences.

[6] E.g. using `awk(1)` or `perl(1)` scripts.
[7] Auto_TSG was developed prior to the database.

It is not possible to have subconstituents in the item: for instance *been being seen* is either one constituent, or several ones: *been* and *being seen* or *been*, *being* and *seen*.

The lexicon looks like this:[8]

```
. PUNCT(.)
a competitor  NP_sg
a manager  NP_sg
about her PP
about the news PP
Abstains V
abstains V
Accounts V
accounts V
```

etc.

The output of the script is the sentence with several annotations filled in (among others positions, instance and category):

```
essex-fouvry@dec-95@invented@formal@none@1@@@Him comes .@0@2@@0:1@
Him@PRON_pers-3-acc-sg-m@@@@@1:2@comes@V@@@@@2:2@.@PUNCT(.)@@@@
essex-fouvry@dec-95@invented@formal@none@1@@@He peers into it .@1@
4@@0:1@He@PRON_pers-3-nom-sg-m@@@@@1:2@peers@V@@@@@2:4@into it@PP@
@@@@4:4@.@PUNCT(.)@@@@
essex-fouvry@dec-95@invented@formal@none@1@@@the manager@1@2@@0:2@
the manager@NP_sg@@@@ essex-fouvry@dec-95@invented@formal@none@1@@
@He has been being seen .@2@5@@0:1@He@PRON_pers-3-nom-sg-m@@@@@1:2
@has@V_aux@@@@@2:5@been being seen@VP_perf@@@@@5:5@.@PUNCT(.)@@@@
```

2.8.2 Tools for consistency checking

All these scripts take a `tsct` *item* file as input. These proved to be a very effective way to check for typographical errors (typically one wrong character, but sometimes that is a field separator, what will have serious consequences for the record in which it occurs).

- `tsct-WF-count` counts the number of well-formed, ill-formed, relatively ill-formed and dubious items in a file, and prints the results on standard output.
- `tsct-cat-list` gives a sorted list of all categories that are used in the input file.
- `tsct-extract` gives a list of all input strings in the input file.
- `tsct-func-list` gives a sorted list of all functions that occur in the input file.

[8]It can be used for lexical replacement using the lexical replacement tool developed in WP5.1 [2].

- `tsct-lex-list` gives a sorted list of all analysis instances that occur in the input file.
- `tsct-test-val` gives the line number and file name (taken from the file names given in the argument) where a certain value occurs. It does not work with regular expressions. The command looks for instance as follows:

  ```
  tsct-test-value -v value="C_Complementation" -v field=3 \
                  C_Complementation.phenomena
  ```

  It prints the filename and the line number of the record where the (here faulty) value occurs. It is designed to find values that were cited in error reports produced by other scripts.

Apart from that, there were also scripts in use handling the ID-numbers, and creating some data files automatically. But as this practice was different for each site, we do not describe these scripts.

These scripts can be obtained from the TSNLP Coordinator. They contain some documentation, but presuppose some knowledge of the programming languages used.

The `cut(1)`, `paste(1)`, `join(1)`, `sort(1)` and `uniq(1)` scripts can be used as well for these purposes, but their command line syntax is rather complicated and there is often a limitation on the number of characters a record can have. The reader is referred to the Unix Manual Pages for more information on these.

Chapter 3

Customisation

3.1 Introduction

The previous chapter on Methodology describes how to construct test suites for core data, i.e. data that is meant to be applicable to a wide range of applications and situations and which contain phenomena that are central to the language. This section considers how to write system-specific data.

TSNLP included a testing phase (see [7]) in which a grammar checker, parser and controlled language checker were tested. This provided the opportunity to investigate how core data could be tuned to a particular system and to test our methodology for test suite construction. On the basis of this we came up with a list of recommendations for customising data which is given in the next section. These recommendations are applicable to any system or application. Further recommendations on how to customise test data for parsers, grammar checkers, and controlled language checkers can be found in [3].

Some of the work involved in customisation can potentially be aided by tools. A prototype lexical replacement tool was developed within TSNLP and is descibed in Volume 2b and in [2].

It was originally intended that the customisation phase of TSNLP would also include the investigation of the tuning of test suites to different corpora, but this task was abandoned when, at the Peer Reviewers' request, we decided to concentrate on the production of more core data. Some observations on this aspect of customisation are discussed in section 3.3.

3.2 A procedure for customisation

Customisation potentially involves the creation of additional annotations which should be written in a file following the database format (see Volume 2) and should incorporate the test item ID number to link it to the rest of the data.

We suggest that in order to customise data to a particular system or application at least the following should be done (not necessarily in the order given):

1. look at system requirements[1]

   (a) determine output attributes
   Output attributes include the system's linguistic output (a string, parse tree, etc), plus any other information one wants to keep track of, such as parse times, which rules have been violated, etc.

   (b) determine form and format of test item
   It is important that the format of the test items be compatible with that allowed by the system. Also, if output annotations are given, then they too should where possible conform to the format produced by the system to enable automatic comparison. (Automatic comparison of test suite annotations with system output is described in [7]). Format includes not just the markup and character set of the test item, but also punctuation conventions.

   As to the form of the test item, this refers to whether the item is a sentence or sub-sentential constituent. Many systems are designed to deal only with sentences, so non-sentential input may cause problems. If this is the case this type of input should be embedded in a simple sentence context.

   (c) determine lexical requirements
   Vocabulary is an important issue and lexical replacement may be needed for the following reasons:

   - a word used in a test item is unknown to the NLP system used for the testing and this is often detrimental to the parsing of the test sentence. Lexical replacement is necessary as the syntactic test suite should not test behaviour in the presence of unknown words.
   - a word used in a test item is syntactically or semantically ambiguous and may generate a different interpretation. Some such ambiguities can be detected already during an initial reading of the test suite where it is possible to adopt a more external and "neutral" point of view on the test items than during the construction phase. Other ambiguities may only be apparent after the test items have been submitted to the system in a round of testing.

2. decide on the phenomena you wish to test
   Various criteria are possible for choosing which phenomena to treat, such as relevance to the particular application or system, or frequency in a certain text type (see section 3.3 below). A customised test suite may need to test for phenomena that lie outside the scope of a general syntactic test suite such as TSNLP. The type of phenomena to treat will depend on the functions of the system. We give some examples of the type of additional phenomena they may need to be covered:

[1] see also Appendix.

(a) Lexicon
While the core TSNLP test suite concentrates on syntactic rather than lexical phenomena, and uses a restricted vocabulary where possible in order to reduce lexical interference, a customised test suite may need to specifically test for lexical phenomena. A controlled language checker, for example, may disallow certain words under any circumstances, or allow its use only in a certain category, or with a certain word sense.

(b) Style
Controlled language checkers (and some grammar checkers) contain rules pertaining to style, so that test items may be required for that too. Style rules may operate at a number of levels (The following examples are from [1]):
Punctuation, e.g.

> *If you start an instruction with a descriptive statement (dependent clause), you must separate that statement from the rest of the instruction with a comma.*

Sentence structure (sentence length, contents, etc), e.g.

> *Write only one instruction per sentence.*

Paragraph structure (e.g. paragraph length, contents), e.g.

> *Each paragraph must have only one topic.*
> *Always start the paragraph with the topic sentence.*

Layout (e.g. headings), e.g.

> *If you want to show that some steps are closely related, use the tabular layout (vertical layout) for the text.*

(c) Error types
While the TSNLP core data was restricted to *competence* errors (ie errors that violate normal grammatical feature restrictions such as the wrong case marking on a noun, or the wrong number on a verb), the data could be extended to cover *performance* errors (ie errors that occur as a result of text manipulation, treatment of formatted structures, punctuation, etc.) for the testing of applications such as grammar checkers. It may also be of interest to categorise errors according to the class of people who made them (to distinguish, for example, second language errors from first language errors).

(d) Ambiguous items
While ambiguity was avoided in the core TSNLP data in order to avoid introducing unnecessary complexity, the testing of ambiguity (categorical and syntactic) may be required for the testing of parsers, for example, to test how a system behaves.

(e) Complex items
While the core data is constructed as simply as possible, complex items can be introduced into the customisation phase. An example of a complex item is a so-called “trap” where patterns which

are relatively easy to check in simple constructions are deliberately complicated. For example, a simple case of subject - verb agreement when the verb follows immediately the head noun of the subject (see example (a) below) can be deliberately complicated by extending the subject with a prepositional complement whose head noun does not agree with the verb in number (see example (b) below). Example (c) below offers an ill-formed, or negative variant of the same trap.

(a) *The cake is on the table.*
(simple test item for subject-verb agreement)

(b) *The cake for the children is on the table.*
(positive trap for subject-verb agreement)

(c) *The cakes for the child is on the table.*
(negative trap for subject-verb agreement)

Traps are useful for testing grammar checkers, for example.

3. create system specific classification of (sub)phenomenon names
During evaluation it may be easier to retrieve items on the basis of the terminology used in the system grammar rather than the general grammatical classification given in TSNLP. The TSNLP classification, however, gives access to initial data collections.

4. establish mapping between categories of the system grammar and the TSNLP categories
This step will be necessary to enable automatic comparison of system output. If this is neither possible nor desirable, TSNLP categories are human-readable.

5. adapt the lexicon of the test suite to that of the system
In order to change the vocabulary of the test suite so that the system to be tested can deal with it we need to ensure the following:

A lexical item which is chosen to replace a word from the original test suite should be a compatible in the sense that it has no other features that could influence the results of the performance. It should have the same syntactic (and in some cases also semantic) features as the word it is replacing.

This is probably best illustrated with some examples.

- When replacing the vocabulary of test sentences for eg NP_Agreement, it is important to choose nouns which distinguish between singular and plural: *this firm* cannot be replaced with *this salmon* as the noun *salmon* can be both plural and singular and thus agreement is not properly tested.
- Possible idiomatic interpretations should also be avoided. For instance, when replacing a verb that takes two arguments, *He carries it.* should not be replaced by *He kicks the bucket.* It is quite possible that the system will deal with the second example in a completely

different way than with the first, and then that particular argument frame would not be tested adequately.

- It is important to check that the (un-)grammaticality of the test item is preserved; for example, when replacing the vocabulary of test sentences containing relative pronouns, it is important not to replace an ungrammatical sentence like **the firms which are working* with *the printers which are working* as the word *printers* can be both human and non-human and therefore the ungrammatical sentence has become grammatical.

All this makes fully automatic lexical replacement very difficult, and necessitates a richly annotated lexicon.

6. expand items if necessary (see step 1)
The augmentation of phrasal items into full sentences must follow the general TSNLP guidelines, ie each phenomenon should be represented in as much isolation as possible in order to minimize potential "noise" from cooccurring phenomena present in the context.

7. write output annotations
Output annotations should be written, observing the formatting conventions mentioned in step 1 above. In order to write output annotations the test suite writer needs access to system output, and where possible, system specifications.

8. write any additional test items of the kind mentioned in step 2 above
The general guidelines of TSNLP core data (see chapter on Methodology) should only be followed here where they are appropriate.

3.3 Tuning test suites to corpora

A test suite is generally accepted as a useful tool for diagnostic evaluation of linguistic rule based systems under development since it is a collection of test data that has been drawn up with the intention to systematically test linguistic phenomena, and which allows a system developer to establish which of the linguistic rules in the system do not work adequately. It is also well-known that it can be useful for other types of system assessment (especially adequacy evaluation) as soon as the evaluator has access to frequency information from typical input texts.

The provision of such information must be seen as one of the most important tasks when customizing a test suite. Information on frequency and weighting relative either to specific corpora or text types plays a major role when assessing the results of applying a test suite to an application.

Some work on aligning test suites to corpora was carried out within the project ([5] and [6]), 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. Such corpus annotations are clearly insufficient as a basis for automatic identification of higher level syntactic structures of the type needed to identify and count examples of eg. subcategorisation frames or types of modifications in corpora. This problem is discussed in [5]. The (semi-)automatic alignment of test suites to corpora remains a topic for future research. In the mean time one can get some idea of relevance and frequency of a phenomenon to some corpus by manual inspection of "representative" samples of that corpus.

Chapter 4

Testing

4.1 Introduction

The previous chapter on customisation describes how to build system-specific data, on the basis of the general purpose TSNLP test suites. The purpose of this chapter is to describe how the test suite can be applied to a real life evaluation scenario.

The testing phase of TSNLP aimed at testing various aspects of the adequacy of the TSNLP test data as diagnostic or evaluation tools, using three different NLP systems, i.e. a grammar checker, a parser and a controlled language checker. Results of investigation on the adequacy of the TSNLP test data and annotations are presented in [7].

In this chapter we will concentrate on reporting the type of information the test suite was able to provide for a specific test scenario - the testing of a commercially available French grammar checker. After a brief description of the evaluation scenario and testing procedure, results of testing will be analysed so as to give an overview of the major characteristics of the system revealed through the use of TSNLP test data and annotations.

4.2 A real life evaluation scenario and testing procedure

The test scenario described below illustrates a type of diagnostic evaluation of a commercially available, general purpose NLP product, carried out in a black box situation.

The spelling and grammar checker/corrector tested was used both in batch and interactive mode. The system provides various types of diagnoses and correction suggestions for each test item, together with details of the morphological and syntactic analysis and a dictionary look up functionality. Three major types of diagnoses are provided by the system: each test item submitted may be considered:

- correct
  Example: "•Je viens", with the following diagnosis: *No error detected.*

- incorrect
  In this case a correction is proposed, with the following diagnosis: *Error detected.*
  Example: "√Tous •**les** hommes" for the input string "Tous la hommes". This correction is explained as follows: *The determiner* **les** *agrees in gender and number with the noun* **hommes** *to which it is related.* For this example, an additional remark is provided: *Warning: the main element of this sentence, the noun* **hommes**, *is not a verb. It is often correct, but it is sometimes an error.*

- impossible to correct
  In this case, the input string could not be entirely analysed and two types of partial diagnoses are provided: the test item is an unknown but "apparently correct" structure, or a structural error has been detected in the input string but cannot be corrected.
  Example 1: "?A •**l'**entreprise/pleut/". The diagnosis of the system is: *Apparently correct original sentence, but I could only partially analyse the sentence.* The structure of the sentence is unknown to the system.
  Example 2: "* Ils lui •**acceptent**". The diagnosis of the system is: *The sentence is not properly structured; manual correction required.* The system detects an error in the structure of the sentence but can neither identify nor correct it.

On the basis of the TSNLP annotations, a selection of four phenomena (complementation, tense_aspect_modality, agreement and word_order) and three phrase levels (Clause, Noun Phrase and Adjectival Phrase) were tested. Three types of TSNLP test items were used : grammatical test items (395), ungrammatical test items (1233) and dubious test items (5), representing a total of 1633 test items.

In practice, the testing consisted in three major phases:

- feeding the test items to the system tested,

- checking the results and diagnoses produced by the system and its performance in the five tasks identified for all grammar checkers, i.e.:

  - assign correct (maybe partial) structure (through parsing or pattern matching) to the input string;
  - identify errors against the grammar;
  - report errors against the grammar;
  - make suggestions for corrections; and
  - not report an error where none exists.

  Due to the nature of TSNLP test data classified in positive and negative test items, this operation mainly consisted in checking that both well-formed and ill-formed test items were treated as such (i.e. that respectively *no error was detected* and *at least one error was detected*).

- identifying and further investigating the cases where the systems grammaticality judgements do not correspond to TSNLP grammaticality judgements.

4.3 The system's performance characteristics revealed through the use of the TSNLP test suites

From a global statistical point of view, the grammar checker produced relatively good results: 67% of the diagnoses provided by the system correspond to the expected output defined in TSNLP. The breakdown of these results is as follows: 96% of well-formed TSNLP test items were considered correct and 58% of ill-formed test items were considered incorrect by the system. However, these results on the system's performance need to be qualified:

- performance differences from one phenomenon to another
  The percentage of correct diagnoses (for well-formed and ill-formed test items) ranged from 97% to 22% depending on the phenomenon tested. The system produced more than 50% correct diagnoses on three phenomena (Clause_Agreement - with the highest rate -, NP_Agreement and NP_Word_Order), whereas four phenomena were generally badly treated by the system (with less than 50% of expected output),
  i.e. Clause_Tense_Aspect_Modality, Complementation in NPs, clauses and APs (with the lowest rate).

- performance differences with well-formed vs ill-formed test items
  TSNLP well-formed test items submitted to the grammar checker were overwhelmingly properly analysed by the system. The system yielded 100% good results when dealing with well-formed test items illustrating the four following phenomena : Clause_Complementation and NP_Complementation, Clause_Tense_Aspect_Modality and NP_Word_Order.

  The system performance changed radically for ill-formed test items, both in terms of global results - only 58% of the system diagnoses were correct - and in terms of phenomena mastered (ranging from 99% to 0% of expected output depending on the phenomenon). A noticeable discrepancy could be observed for four phenomena (Clause_Complementation and NP_Complementation, Clause_Tense_Aspect_Modality and NP_Word_Order) where the system yielded very good results on well-formed TIs (100%) and very bad ones on ill-formed test items (from 35% to 6%).

The detailed results are presented in the table below.

French Phenomenon	Total number of TSNLP TIs	Total system correct diagnoses	system correct diagnoses for *well formed* TIs (%)	system correct diagnoses for *ill formed* TIs (%)	Total % of system correct diagnoses
Clause_Agreement	163	158	93%	99%	97%
NP_Agreement	677	626	93%	92%	92%
NP_Word_Order	60	31	100%	6%	52%
Clause_Tense_Aspect_Modality	348	170	100%	35%	49%
NP_Complementation	20	8	100%	14%	40%
Clause_Complementation	333	98	100%	7%	29%
AP_Complementation	32	7	78%	0%	22%
TOTAL	1633	1098	96%	58%	67%

Distribution of correct diagnoses

This first round of results showed that the grammar checker performed much better when dealing with well-formed test data than with ill-formed ones. This observation also needs to be qualified by a more detailed analysis of the types of incorrect or unexpected diagnoses produced by the system. The breakdown of these results is presented in the table below.

French Phenomenon	Total number of TSNLP TIs	Total of system incorrect or unexpected diagnoses		Total % of system incorrect or unexpected diagnoses
		items incorrectly treated by the system	items not analysed by the system	
AP_Complementation	32	11	14	78%
Clause_Complementation	333	118	117	71%
NP_Complementation	20	8	4	60%
Clause_Tense_Aspect_Modality	348	24	154	51%
NP_Word_Order	60	5	24	48%
NP_Agreement	677	47	4	8%
Clause_Agreement	163	3	2	3%
TOTAL	1633	216	319	33%

Distribution of incorrect or unexpected diagnoses

Three types of unexpected output were produced by the system:

- well-formed test items were considered incorrect by the system,
- ill-formed test items were considered correct by the system,
- both well-formed and ill-formed items did not generate any precise diagnosis from the system.

When analysing the corrections proposed by the system, a distinction should be made between:

- the incorrect diagnoses produced by the system, which revealed the system's lexical or morpho-syntactic deficiencies; and
- the unexpected output produced by the system, which did not correspond to the TSNLP positive test data but were however plausible.

A few cases have also been observed where the system produced two possible diagnoses, and only one was correct.

For well-formed test items, the percentage of incorrect diagnoses was relatively low (4%) and was limited to three phenomena: AP_Complementation (2 TIs not fully analysed), Clause_Agreement (3 TIs considered incorrect and 1 not fully analysed) and NP_Agreement (10 TIs considered incorrect). If we concentrate on appropriate TSNLP well-formed test data, with no lexical or syntactic ambiguities, different cases of system output can be identified:

1. deficiencies of the system

   - The system could not analyse a well-formed test item illustrating the Clause_Agreement phenomenon

TSNLP input	System Output
Tu viens	?Tu/viens/

   - The system could not analyse well-formed isolated adjectival phrases

TSNLP input	System Output
heureux que √**l'**ingénieur vienne	? heureux que l'ingénieur vienne/
difficile de comprendre	? difficile/de comprendre/.

   - The system provided two diagnoses: only one was correct

TSNLP input	System Output
La meilleure question	√•Là meilleure question •La meilleure question

2. unexpected but plausible system output

   - agreement of coordinated adjectives in isolated NPs

TSNLP input	System Output
Deux cérémonies civile et religieuse	√Deux cérémonies •civiles et •religieuses

   - Clause_Agreement

TSNLP input	System Output
La plupart d'entre nous vient	√La plupart d'entre nous •**viennent**.
Ce sont les hommes	√ •**C'**•**est** •**l'** •**homme**.

For ill-formed test items, the percentage of incorrect diagnoses was much higher (42%) and concerned all the phenomena tested, with a strong rate of bad results for AP_Complementation (100%), NP_word_Order (94%), Clause_Complementation (93%) and NP_Complementation (86%).

The major deficiencies of the system revealed through the use of ill-formed test data were the following:

- The system could not detect ill-formed isolated adjectival phrases

TSNLP input	System Output
*jaloux à l' ingénieur *difficile de l' entreprise pour comprendre	•jaloux à l' ingénieur •difficile de l'entreprise pour comprendre

- The system could not detect errors in some ill-formed test items illustrating the phenomena of Clause_Complementation, Clause_Tense_Aspect_Modality, NP_Agreement, NP_Complementation and NP_Word_Order.

French phenomenon	Examples of ill-formed test items considered correct by the system
Clause_Complementation	*Il neige la question.
Clause_Tense_Aspect_Modality	*Nous/Vous travaillé
NP_Agreement	*Les meilleurs travail
NP_Complementation	*La cause à la rupture
NP_Word_Order	*travail un/travail aucun

- The system could not assign correct structure to some ill-formed test items illustrating the phenomena of complementation, agreement, tense_aspect_modality and word_order.

French phenomenon	Examples of ill-formed test items not fully analysed by the system
AP_Complementation	*attentif avec la question
Clause_Agreement	*Tu vient .
Clause_Complementation	*Il leur rêve.
Clause_Tense_Aspect_Modality	*J'ai travaillant
NP_Agreement	*le/les plupart
NP_Complementation	*La crainte combattre
NP_Word_Order	*Le habituel travail et intelligent

In view of these results, the major characteristics of the grammar checker which have been revealed through the use of the TSNLP test data and annotations can be summarised as follows:

- TSNLP ill-formed test items are frequently not detected as such (more than 16% of them are considered correct by the system).
- The system produces very good results on (both well-formed and ill-formed) test items illustrating the phenomenon of agreement, in clauses (97% of correct diagnoses) and in noun phrases (92%).
- The system does not master the phenomenon of complementation, especially in adjectival phrases (78% of incorrect output).

- Sentential test items produce better results than isolated ones. Punctuation (i.e. the presence or absence of a full stop at the end of a test item) also plays a role in the system performance.

- Many grammatical structures are unknown to the system (the system cannot assign any structure to 17% of TSNLP input strings, including 3 well-formed and 276 ill-formed test items). Structural but unidentified errors have been detected in 40 ill-formed test items illustrating the phenomenon of Clause_Complementation.

The TSNLP data and annotations also enabled us to learn more about the systems performance, in terms of systematicity, lexical and morpho-syntactic deficiencies, and interference with other system components :

- systematicity
  The system produces inconsistent results for a clearly defined phenomenon, depending on the status of the test items used (well-formed vs ill-formed). This characteristic is very important for the evaluator since reliable information on the system performance cannot be derived by using only positive or negative test items for one particular phenomenon.

- lexical deficiency
  The encodings of the lexical entries in the system dictionary may influence the results produced by the system. Access to the system dictionary is therefore essential for the evaluator to interpret unexpected system output. For example, test items illustrating the phenomenon of determiner-noun agreement are generally properly treated by the system, except when they contain the word *plupart*, since *la plupart* is encoded as a single lexical entry, classified as a 3rd person singular pronoun. Similarly, for the phenomenon of complementation in noun phrases, test items built with nouns designating quantities produce correct results, except when they contain the word *personne* which is categorially ambiguous between noun and pronoun.

- morpho-syntactic deficiency
  For the phenomenon of agreement in noun phrases, the testing shows that the system can properly deal with noun/adjective agreement, except when the adjective used in the test item is an irregular superlative. Quantifiers used in complex determiners embedded in noun phrases systematically produce incorrect output. The phenomenon of subject-verb agreement is properly treated by the system when the verb is inflected for person and number but the system behaviour is inconsistent if the ill-formedness of a test item consists in using an infinitive instead of a finite verb form. The poor results produced by the system when dealing with Clause_Complementation comes from the fact that the subcategorisation information provided in the dictionary entries is not restrictive enough so that incorrect syntactic variations are allowed for the "object" of transitive verbs.

- interference of phonetic and graphical rules with syntactic knowledge Some unexpected, though plausible, diagnoses produced by the system show that spelling rules sometimes prevail over syntactic rules in the correction process. For example, when correcting the ill-formed test item **Je travailler.* by *Je travaillai.* instead of the expected TSNLP well-formed *Je travaille.*, the system gives precedence to homophony rules. Similarly, when correcting the ill-formed test item **La vient* by *La vie.* instead of the expected TSNLP well-formed *Elle vient.*, the system gives precedence to homography rules. In such cases, customisation would be needed to create test items dedicated to grammar checkers, consisting in phonetically and graphically non ambiguous lexical items, which would avoid interference with the spelling checker and allow the testing of the syntactic correction only.

In this real life experience, TSNLP test data and annotations provide reliable and valuable information on the performance of a commercially available NLP product. The results of this testing also highlight the necessity to build additional, system-specific customised test data, in any particular evaluation situation. The TSNLP annotation schema and tools have been developed to permit such extensions.

Appendix A

Checklist of System requirements

The following checklist is not intended to be exhaustive, but indicative of the kind of thing a test suite builder should consider when customising test suites.

- What information does the system give the user/tester?
  - linguistic information:
    Parse trees or other forms of markup, corrections of unacceptable input, suggestions, warnings, etc.
  - non linguistic information:
    Parse times, error messages (system internal), references to the grammars of the system and/or the system specifications, etc.

  All these can be used as attributes for an output template, in addition to author, date, revisions, ID-number in the existing TSNLP test suite.

- What system specific input is needed?
  Adapted punctuation and other conventions (like SGML markup), special ID-numbers (if the general ones cannot be used), etc.

- list of the phenomena and description of phenomena (system specifications and implemented grammars)

Bibliography

[1] AECMA/AIA Simplified English

[2] Arnold, Doug, Martin Rondell & Frederik Fouvry. (1994). *Design and Implementation of Test Suite Tools*, Report to LRE 62-089 (D-WP5.1), University of Essex.

[3] Balkan, Lorna, Siety Meijer et al. (1994). *Test Suite Design — Guidelines and Methodology*, Report to LRE 62-089 (D-WP2.1a), University of Essex.

[4] Balkan, Lorna, Siety Meijer et al. (1994). *Issues in Test Suite Design*, Report to LRE 62-089 (D-WP2.1b), University of Essex.

[5] Balkan, L. and Fouvry, F. (1995). *Corpus-based test suite generation*, Report to LRE 62-089, University of Essex.

[6] Dauphin, E., Lux, V., Regnier-Prost, S. et al. (1995). *Checking against Corpora*, Report to LRE 62-089.

[7] Dauphin, E., Lux, V., Regnier-Prost et al. (1995). *Testing and Customisation of Test Items* Report to LRE 62-089 (D_WP4.2&3).

[8] Duden 1984, *Grammatik der deutschen Gegenwartssprache*, Dudenverlag, Bibliographisches Institut MannheimWienZürich.

[9] EAGLES Evaluation Subgroup. (1994): *Evaluation of Natural Language Processing Systems (Draft)*, EAG-EWG-PR.2, Centre for Language Technology, Copenhagen, Denmark.

[10] Estival, Falkedal et al. (1994), *Analysis of Existing Test Suites*, Report to LRE 62-089 (D-WP1), University of Essex.

[11] Grevisse, M. *Le bon usage*, Éditions Duculot, Paris - Louvain-la-Neuve, 1986

[12] Quirk, R., Greenbaum, S., Leech, G., Svartvik, J. *A Comprehensive Grammar of the English Language*, Longman, London 1985